UNCLASSIFIED

AD NUMBER ADB120256 LIMITATION CHANGES TO: Approved for public release; distribution is unlimited. FROM: Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; JUL 1987. Other requests shall be referred to Air Force Armament Lab., Eglin AFB, FL. This document contains export-controlled technical data. AUTHORITY AFSC/MNOL wright lab ltr dtd 13 Feb 1992

256
0
12
m
Ţ
<

SECURITY CLASSIFICATION OF THIS PAGE					
REPORT (OCUMENTATIO	N PAGE			Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION		16. RESTRICTIVE	MARKINGS	-	
Unclassified					
2a. SECURITY CLASSIFICATION AUTHORITY			/AVAILABILITY OF		C Consemment
2b. DECLASSIFICATION / DOWNGRADING SCHEDU	LE	Agencies	and their co	ontracto	S. Government ors; CT (over)
4. PERFORMING ORGANIZATION REPORT NUMBE	R(S)		ORGANIZATION RI R-88-18, Vo		MBER(S)
6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL	7a. NAME OF M	ONITORING ORGAI	NIZATION	
McDonnell Douglas	(If applicable)	Aeromech	anics Division	on	
Astronautics Company					
6c. ADDRESS (City, State, and ZIP Code)		76. ADDRESS (Cit	y, State, and ZIP (Armament)	lode) Laborat	orv
P.O. Box 516 St. Louis, MO 63166		Eglin AFI	3, FL 32542	-5434	<u>.</u>
8a. NAME OF FUNDING / SPONSORING	86. OFFICE SYMBOL	9. PROCUREMEN	T INSTRUMENT ID	ENTIFICATI	ON NUMBER
ORGANIZATION STARS Joint Program Office	(If applicable)	F08635-8	6-C-0025		
8c. ADDRESS (City, State, and ZIP Code) Room 3D139 (1211 Fern St)		10. SOURCE OF F	UNDING NUMBER	S	i -
The Pentagon		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO. 5
Washington DC 20301-3081		63756D	921C	GZ	57
11. TITLE (Include Security Classification)		Q3130D	9210	GZ	
Common Ada Missile Package Detail Design Documents (Vol	(CAMP) Project: 7-12)	Missile Soft	ware Parts,	Vol 9:	
12. PERSONAL AUTHOR(S)		<u>-</u>			
D. McNicholl, S. Cohen, C. I	Palmer, et al.	14. DATE OF REPO	OT /Vone Month	Onul 115	PAGE COUNT
Technical Note FROM S	OVERED ep 85 _{TO} Mar 88:	March 198		Day, 13.	422
16 CLIPPI FACALTARY NOTATION	ECT TO EXPOR				
	his report is spe			cover.	(over)
17. COSATI CODES	18. SUBJECT TERMS (C	ontinue on revers	e if necessary and	identify b	by block number) Ware G enerators
↑ FIELD GROUP SUB-GROUP		Composition			
	11111				
19. WBSTRACT (Continue on reverse if necessary	and identify by block no	umber)	foogibility 0	f rough	ble Ada software
The objective of the CAMP pr parts in a real-time embedded	ogram is to demo	nstrate the	n chosen for	the de	emonstration was
that of missile flight software	systems. This	required that	at the existe	nce of	commonality
within that domain be verified	i (in order to ju:	stify the dev	relopment of	parts	for that domain),
and that software parts be de	esigned which ad	dress those	areas identi	fied.	An associated
parts system was developed t Guide to the CAMP Software.	o support parts	usage. Von	ime I of this	on Does	ument. Volume 3
is the Software Product Speci	fication: Volume	s 4-6 contain	n the Top-L	evel De	sign Document:
and, Volumes 7-12 contain th	e Detail Design I	Documents.			DTIC
Some		✓		_	
	0				ADD 0 7 4000
					APR 0 7 1988
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED AS F	IPT. DTIC USERS	21. ABSTRACT SE Unclassif	CURITY CLASSIFIC	ATION T	S. C.
22a. NAME OF RESPONSIBLE INDIVIDUAL			Include Area Code -2961) 22c. OF	FICE SYMBOL
Christine Anderson		1 (003) 002	2001	1 44	

UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

this report decuments test and evaluation; distribution limitation applied March 1988. Other requests for this document must be referred to AFATL/FXG, Eglin AFB, # Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

These technical notes accompany the CAMP final report AFATL-TR-85-93 (3 Vols)

AFATL-TR-88-18, Vol 9

SOFTWARE DETAILED DESIGN DOCUMENT

FOR THE

MISSILE SOFTWARE PARTS

OF THE

COMMON ADA MISSILE PACKAGE (CAMP) PROJECT

CONTRACT F08635-86-C-0025

CDRL SEQUENCE NO. C007



-4		
Acces	sion For	
NTIS	GRA&I	Q.
DTIC	TAB	X I
Unann	ounced	
Justi	fication_	
By		
Distribution/		
Avai	lability	Codes
	Avail an	d/or
Dist	Specia	1
1	1 12	1
		and the second
P-2		W/V

30 OCTOBER 1987

Distribution authorized to U.S. Government agencies and their contractors only; this report documents test and evaluation; distribution limitation applied July 1987. Other requests for this document must be referred to the Air Force Armament Laboratory (FXG) Eglin Air Force Base, Florida 32542 – 5434.

<u>DESTRUCTION NOTICE</u> – For classified documents, follow the procedures in DoD 5220.22 – M, Industrial Security Manual, Section II – 19 or DoD 5200.1 – R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

<u>WARNING:</u> This document contains technical data whose export is restricted by the Arms Export Control Act (Title 22, U.S.C., Sec. 2751, et seq.) or the Export Admin – istration Act of 1979, as amended (Title 50, U.S.C., App. 2401, et seq.). Violations of these export laws are subject to severe criminal penalties. Disseminate in accordance with the provisions of AFR 80 – 34.

AIR FORCE ARMAMENT LABORATORY

Air Force Systems Command United States Air Force Eglin Air Force Base, Florida

89 4 6 128



3.3.6.2 GENERAL VECTOR MATRIX ALGEBRA (BODY) TLCSC P682 (CATALOG #P197-0)

This part is a package of generic packages and generic functions. The LLCSC's take two different forms. One form defines vector and matrix types, along with general operations on these types. The other form requires that vector and matrix types be provided as generic parameters and performs operations on data objects of different types.

Many of the parts have both an unconstrained and constrained or restricted and unrestricted versions. The constrained/restricted versions of these parts are less flexible in the dimensioning of the input arrays, but require fewer internal calculations.

The generic functions/package which import generic formal array types have been designed to work in conjunction with the data types exported by the generic packages.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:



Name	Requirements Allocation
General Vector Matrix Algebra	I RO58 I
Vector Operations Unconstrained	R061, R062, R063,
	R104
Vector_Operations_Constrained	RO61, RO62, RO63,
	R104
Matrix Operations Unconstrained	RO75, RO76, RO79,
<u> </u>	R080, R155, R156
Matrix Operations Constrained	RO75, RO76, RO79,
	R080, R155, R156
Dynamically Sparse Matrix Operations Unconstrained	R226
Dynamically Sparse Matrix Operations Constrained	R226
Symmetric Half Storage Matrix Operations	R211
Symmetric Full Storage Matrix Operations Unconstrained	R227
Symmetric Full Storage Matrix Operations Constrained	R227
Diagonal Matrix Operations	R212
Vector_Scalar_Operations_Unconstrained	RO65, RO66
Vector_Scalar_Operations_Constrained	R065, R066
Matrix_Scalar_Operations_Unconstrained	R073, R074
Matrix_Scalar_Operations_Constrained	R073, R074
Diagonal Matrix Scalar Operations	R212
Matrix_Vector_Multiply_Unrestricted	R069
Matrix_Vector_Multiply_Restricted	R069
Vector_Matrix_Multiply_Unrestricted	N/A
Vector Matrix Multiply Restricted	N/A
Vector Vector Transpose Multiply Unrestricted	N/A
Vector Vector Transpose Multiply Restricted	N/A
Matrix_Matrix_Multiply_Unrestricted Matrix_Matrix_Multiply_Restricted	R077 R077
Matrix Matrix Transpose Multiply Unrestricted	N/A
Matrix Matrix Transpose Multiply Restricted	N/A
Dot Product Operation Unrestricted	R063
Dot Product Operation Restricted	R063
Diagonal Full Matrix Add Unrestricted	R212
Diagonal Full Matrix Add Restricted	R212
ABA Trans Dynam Sparse Matrix Sq Matrix	N/A
ABA_Trans_Vector_Sq_Matrix	N/A
ABA Trans Vector Scalar	N/A
ABA Trans Col Matrix Sq Matrix	N/A
Column Matrix Operations	N/A
1	

3.3.6.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.3 INPUT/OUTPUT

None.



3.3.6.2.4 LOCAL DATA

None.

3.3.6.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.6 PROCESSING The following describes the processing performed by this part: package body General_Vector_Matrix_Algebra is package body Vector Operations Unconstrained is separate; package body Vector Operations Constrained is separate; package body Matrix Operations Unconstrained is separate; package body Matrix Operations Constrained is separate; package body Dynamically Sparse Matrix Operations Unconstrained is separate; package body Dynamically Sparse Matrix Operations Constrained is separate; package body Symmetric Half Storage Matrix Operations is separate; package body Symmetric Full Storage Matrix Operations Unconstrained is separate; package body Symmetric Full Storage Matrix Operations Constrained is separate; package body Diagonal Matrix Operations is separate; package body Vector Scalar Operations Unconstrained is separate; package body Vector Scalar Operations Constrained is separate; package body Matrix Scalar Operations Unconstrained is separate; package body Matrix Scalar Operations Constrained is separate; package body Diagonal Matrix Scalar Operations is separate; package body Matrix Vector Multiply Unrestricted is separate; function Matrix Vector Multiply Restricted (Matrix : Input Matrices; Vector : Input Vectors) return Output Vectors is separate; package body Vector_Vector_Transpose_Multiply Unrestricted is separate;



function Vector Vector Transpose Multiply Restricted : Left Vectors ;

Right: Right Vectors) return Matrices is separate;

None.

```
package body Matrix Matrix Multiply Unrestricted is separate;
   function Matrix Matrix Multiply Restricted
               (Left : Left Matrices;
                Right : Right Matrices) return Output Matrices is separate;
   package body Matrix Matrix Transpose Multiply Unrestricted is separate;
   function Matrix Matrix Transpose Multiply Restricted
               (Left : Left Matrices:
                Right: Right Matrices) return Output Matrices is separate;
   package body Dot Product Operations Unrestricted is separate;
   function Dot Product Operations Restricted
               (Left : Left Vectors;
                Right : Right Vectors)
               return Result Elements is separate;
   package body Diagonal Full Matrix Add Unrestricted is separate;
   function Diagonal Full Matrix Add Restricted
               (D Matrix : Diagonal Matrices;
                F Matrix: Full Matrices) return Full Matrices is separate;
  package body Vector_Matrix_Multiply_Unrestricted is separate;
   function Vector Matrix Multiply Restricted
               (Vector : Input Vectors;
                Matrix : Input Matrices) return Output Vectors is separate;
  package body ABA Trans Dynam Sparse Matrix Sq Matrix is separate;
  package body ABA Trans Vector Sq Matrix is separate;
  package body ABA Trans Vector Scalar is separate;
  package body Column Matrix Operations is separate;
end General Vector Matrix Algebra;
3.3.6.2.7 UTILIZATION OF OTHER ELEMENTS
None.
3.3.6.2.8 LIMITATIONS
```



3.3.6.2.9 LLCSC DESIGN

3.3.6.2.9.1 VECTOR_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P337-0)

This package contains functions which provide a set of standard vector operations. The operations provided are addition, subtraction, and dot product of like vectors, along with a vector length operation.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.1.1 REQUIREMENTS ALLOCATION

The following table describes the allowing of requirements to this part:

	Requirements
Name	Allocation
Dot_Product Vector_Length	R063 R104 R061 R062



3.3.6.2.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were defined at the package specification level:

Data types:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector Elements	floating	Resulting type from the operation
Squared	point type	Vector Elements * Vector Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type



Name	Type	Description
11*11	function	Used to define the operation
(Vector_Elements * Vector_Elements :=
SqRt	function	Square root function taking an object of type Vector Elements Squared and returning an object
		of type Vector_Elements

3.3.6.2.9.1.4 LOCAL DATA

Data types:

The following table summarizes the types defined in this part's specification:

Name	Range	Description
Vectors	N/A	Unconstrained, one-dimensional array of elements

3.3.6.2.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Vector_Operations_Unconstrained is

end Vector_Operations_Unconstrained;

3.3.6.2.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.1.8 LIMITATIONS

None.

3.3.6.2.9.1.9 LLCSC DESIGN

None.



3.3.6.2.9.1.10 UNIT DESIGN

3.3.6.2.9.1.10.1 "+" (VECTOR + VECTORS := VECTORS) UNIT DESIGN (CATALOG #P338-0)

This function adds two vectors by adding each of the individual elements in the input vector, returning the resultant vector. All three vectors are of the same type. If the two input vectors do not have the same length, the exception DIMENSION_ERROR is raised. The ranges of the dimensions of the input vectors do not have to be the same.

3.3.6.2.9.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO61.

3.3.6.2.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	Type	Mode	Description	<u>-</u>
	Left Right	Vectors Vectors	In In	One of the vectors to be added Second vector to be added	

3.3.6.2.9.1.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name	1	Type	Ī	Description
- [L Index	1	Indices	1	Vector being calculated and returned Index into Left vector Index into Right vector

3.3.6.2.9.1.10.1.5 PROCESS CONTROL

Not applicable.



```
3.3.6.2.9.1.10.1.6 PROCESSING
The following describes the processing performed by this part:
   function "+" (Left : Vectors;
                 Right: Vectors) return Vectors is
      --declaration section-
      Answer : Vectors(Left'RANGE);
      L_Index : Indices;
      R Index : Indices;
-- --begin function "+"
__ ____
   begin
      --make sure lengths of input vectors are the same
      if Left'LENGTH = Right'LENGTH then
         L Index := Left'FIRST;
         R Index := Right'FIRST;
         Process:
            loop
               Answer(L Index) := Left(L Index) + Right(R Index);
               exit Process when L Index = Left'LAST;
               L Index := Indices'SUCC(L Index);
              R Index := Indices'SUCC(R Index);
            end loop Process;
     else
         --dimensions of vectors are incompatible
         raise Dimension Error;
     end if;
      return Answer;
   end "+";
3.3.6.2.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
```



The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and are defined in the package specification for Vector Operations Unconstrained:

Name	Type	Description
Vector_Elements Indices	floating point type discrete type	Type of elements to be contained in vector type defined by this package Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector Operations Unconstrained:

	Name	Range	Description
	Vectors	N/A	Unconstrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Ī	Name	1	Туре	Ī	Description]
	dimension_error		exception		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.1.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name		When/V	/hy	Rai	sed								
Dimension_Error		Raised same	if	the	lengths	of	the	input	vectors	are	not	the	



3.3.6.2.9.1.10.2 "-" (VECTORS - VECTORS := VECTORS) UNIT DESIGN (CATALOG #P339-0)

This part subtracts one vector from another by subtracting the individual elements of each input vector, returning the resultant vector. The dimensions of the two input vectors must have the same length, but are not required to have the same range.

3.3.6.2.9.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R062.

3.3.6.2.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	1	Туре	1	Mode	Ī	Description	
			Vectors Vectors				Vector to act as the minuend Vector to act as the subtrahend	

3.3.6.2.9.1.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Туре	Description
L Index	Indices	Vector being calculated and returned Index into Left vector Index into Right vector

3.3.6.2.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

function "-" (Left : Vectors:

```
Right: Vectors) return Vectors is
      --declaration section-
      Answer : Vectors(Left'RANGE);
      L Index : Indices;
      R Index : Indices;
-- --begin function "-"
   begin
      -- make sure lengths of the input vectors are the same
      if Left'LENGTH = Right'LENGTH then
         L Index := Left'FIRST;
         R Index := Right'FIRST;
         Process:
            loop
               Answer(L Index) := Left(L Index) - Right(R Index);
               exit Process when L Index = Left'LAST;
               L Index := Indices'SUCC(L Index);
               R Index := Indices'SUCC(R Index);
            end loop Process;
      else
         --dimensions of vectors are incompatible
         raise Dimension_Error;
      end if;
      return Answer;
   end "-";
3.3.6.2.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
The following tables describe the elements defined in this part's top level
component and used by this part:
Data types:
```

The following generic types are available to this part and defined at the package specification level for Vector Operations Unconstrained:

Name	Type	Description
Vector_Elements	floating point type	
Indices 	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector Operations Unconstrained:

1	Name	Range	Description	
	Vectors	N/A	Unconstrained, one-dimensional array of elements	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector Matrix Algebra:

-	Name	 	Туре		Description	
	dimension_error		exception		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.1.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	1	When/Why	Rai	sed								I
]	Dimension_Error		Raised if same	the	lengths	of	the	input	vectors	are	not	the	

3.3.6.2.9.1.10.3 VECTOR LENGTH UNIT DESIGN (CATALOG #1/340-0)

This function calculates the length of a vector, returning the result. The length of a vector is defined as:

a := Sqrt(sum b(i)**2)



3.3.6.2.9.1.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R104.

3.3.6.2.9.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.1.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	-	Туре	1	Mode	1	Descri	ipti	on				1
1	Input	1	Vectors	1	In	1	Vector	for	which a	length	is	desired	I

3.3.6.2.9.1.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Type	Description	
Temp	Vector	_Elements_Squared Used for intermediate calculations	

3.3.6.2.9.1.10.3.5 PROCESS CONTROL

Not applicable.

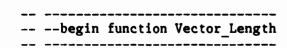
3.3.6.2.9.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

function Vector_Length (Input : Vectors) return Vector_Elements is

-- --declaration section-

Temp : Vector Elements Squared;





```
begin
```

3.3.6.2.9.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector Operations Unconstrained:

Name	Туре	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector Elements * Vector Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Unconstrained:

Ī	Name	1	Range		Description
	Vectors		N/A		Unconstrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General Vector Matrix Algebra:

1	Name	Ī	Туре		Mode		Description
	Left		Vectors		In		First vector to be used in the dot product operation
İ	Right	İ	Vectors	İ	In	İ	Second vector to be used in the dot product operation

3.3.6.2.9.1.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name		Туре		Description	<u></u>
	L Index	Т	Vector_Elements_Squared Indices Indices	- 1	Result of the dot product operation Index into Left vector Index into Right vector	

3.3.6.2.9.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

function Dot Product (Left : Vectors;

-- -- make sure lengths of the input vectors are the same if Left'LENGTH = Right'LENGTH then

```
Answer := 0.0;
L_Index := Left'FIRST;
R_Index := Right'FIRST;
```



```
Process:
    loop

Answer := Answer + Left(L_Index) * Right(R_Index);

exit Process when L_Index = Left'LAST;

L_Index := Indices'SUCC(L_Index);

R_Index := Indices'SUCC(R_Index);

end loop Process;

else

--dimensions of vectors are incompatible raise Dimension_Error;

end if;

return Answer;

end Dot_Product;
```

3.3.6.2.9.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Unconstrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector Elements * Vector Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Unconstrained:



.

Name	Range	Description
Vectors	N/A	Unconstrained, one-dimensional array of elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

1	Name	1	Туре		Description
	dimension_error		exception		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

Subprograms:

The following table summarizes the generic subroutines available to the part and defined at the package specification level for Vector Operations:

1	Name	Туре	Description
	n×n	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared

3.3.6.2.9.1.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

1	Name		When/Why	Rai	sed								
	Dimension_Error		Raised if the same	the	lengths	of	the	two	input	vectors	are	not	

3.3.6.2.9.2 MATRIX OPERATIONS UNCONSTRAINED PACKAGE DESIGN (CATALOG #P347-0)

This package contains subroutines which provide a set of standard operations on matrices of like types.

The decomposition for this part is the same as that shown in the Top-Level Design Document.



3.3.6.2.9.2.1 REQUIREMENTS ALLOCATION

This following illustrates the allocation of requirements to the units in this package.

 Name	Requirements Allocation	
"+" (matrices + matrices) "-" (matrices - matrices) "+" (matrices + elements) "-" (matrices - elements) Set_to_Identity_Matrix Set_to_Zero_Matrix "*"	R079 R080 R075 R076 R155 R156 R077	

3.3.6.2.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.3 INPUT/OUTPUT



GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Ī	Name	Туре	Description	. <u></u>
I	Elements	floating point type	Used to define type of elements in matrix defined by this package	
İ	Col_Indices	discrete type	Used to define second dimension of exported matrix type	Ì
İ	Row_Indices	discrete type	Used to define first dimension of exported matrix type	

3.3.6.2.9.2.4 LOCAL DATA

Data types:

The following data type was previously defined at the package specification level:



Name	Ī	Range		Description	1
Matrices		N/A		Unconstrained, two-dimensional array of Elements	

3.3.6.2.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.6 PROCESSING

The following describes the processing performed by this part:

separate (General Vector Matrix Algebra)
package body Matrix Operations Unconstrained is

end Matrix_Operations_Unconstrained;

3.3.6.2.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.2.8 LIMITATIONS

None.

3.3.6.2.9.2.9 LLCSC DESIGN

None.

3.3.6.2.9.2.10 UNIT DESIGN

3.3.6.2.9.2.10.1 "+" (MATRICES + MATRICES := MATRICES) UNIT DESIGN (CATALOG #P348-0)

This function adds two matrices by adding the individual elements of each input matrix, returning the resultant matrix. The lengths of the first dimensions of the input matrices must be equal, as must be the lengths of the second dimensions. None of the ranges for the dimensions have to be the same.

3.3.6.2.9.2.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R079.



3.3.6.2.9.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be added
Right	Matrices	In	Second matrix to be added

3.3.6.2.9.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Туре	Description
Answer L_Col L_Row R_Col R_Row	Matrices Col_Indices Row_Indices Col_Indices Row_Indices	Result of adding the two input matrices Left column index Left row index Right column index Right row index

3.3.6.2.9.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

function "+" (Left : Matrices:

Right: Matrices) return Matrices is

-- --declaration section-

Answer : Matrices(Left'RANGE(1), Left'RANGE(2));

L_Col : Col_Indices; L_Row : Row Indices; R_Col : Col_Indices; R_Row : Row_Indices;



```
-- --begin function "+"
   begin
      -- make sure the dimensions of the matrices are compatible
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then
         L Row := Left'FIRST(1);
         R Row := Right'FIRST(1);
         Row Loop:
            Toop
               L Col := Left'FIRST(2);
               R Col := Right'FIRST(2);
               Col Loop:
                  Toop
                      Answer(L_Row, L_Col) := Left(L_Row, L_Col) +
                                               Right(\overline{R} Row, \overline{R} Col);
                      exit Col Loop when L Col = Left'LAST(2);
                      L Col := Col Indices SUCC(L Col);
                      R Col := Col Indices'SUCC(R Col);
                  end loop Col Loop;
               exit Row Loop when L Row = Left'LAST(1);
               L Row := Row Indices SUCC(L Row);
               R_Row := Row_Indices'SUCC(R_Row);
            end loop Row Loop;
      else
         -- input matrices have incompatible dimensions
         raise Dimension Error;
      end if;
      return Answer;
   end "+";
3.3.6.2.9.2.10.1.7 UTILIZATION OF OTHER ELEMENTS
```

The following tables describe the elements defined in this part's ancestral components and used by this part:

UTILIZATION OF ANCESTRAL ELEMENTS:



Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Ī	Name	Туре	Description
Ī	Elements	floating point type	Used to define type of elements in matrix defined by this package
Ì	Col_Indices	discrete type	Used to define second dimension of exported matrix type
İ	Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Unconstrained:

1	Name		Range		Description	
	Matrices	1	N/A		Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

Name		Description	<u>-</u>
dimensi	on_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.2.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	1	When/Why R	Raised	-
	Dimension_Error		Raised if t dimensions	the respective lengths of the first and second s of the input matrices are not equal	

3.3.6.2.9.2.10.2 "-" (MATRICES - MATRICES := MATRICES) UNIT DESIGN (CATALOG #P349-0)

This function subtracts one matrix from another by subtracting the individual elements of the input matrices, returning the resultant matrix. The lengths of



the first dimensions of the input matrices must be equal, as must be the lengths of the second dimensions. None of the ranges for the dimensions have to be the same.

3.3.6.2.9.2.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R076.

3.3.6.2.9.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	 	• •	•	•	Description	-
	Left Right		Matrices Matrices		 	Matrix to act as the minuend Matrix to be used as the subtrahend	

3.3.6.2.9.2.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name	Type	Description	
	Answer L_Col L_Row R_Col R_Row	Matrices Col_Indices Row_Indices Col_Indices Row_Indices	Result of adding the two input matrices Left column index Left row index Right column index Right row index	

3.3.6.2.9.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

function "-" (Left : Matrices;

Right: Matrices) return Matrices is

```
3
```

```
--declaration section-
      -----
      Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
      L_Col : Col_Indices;
L_Row : Row_Indices;
R_Col : Col_Indices;
R_Row : Row_Indices;
-- --begin function "-"
  begin
      --make sure matrix dimensions are compatible
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then
         L Row := Left'FIRST(1);
         R Row := Right'FIRST(1);
         Row Loop:
            Toop
                L_Col := left'FIRST(2);
               R Col := Right'FIRST(2);
               Col Loop:
                  Гоор
                      answer(L Row, L Col) := Left(L Row, L Col) -
                                                Right(R Row, R Col);
                      exit Col_Loop when L_Col = Left'LAST(2);
                      L Col := Col Indices'SUCC(L Col);
                      R Col := Col Indices'SUCC(R Col);
                   end loop Col_Loop;
               exit Row Loop when L Row = Left'LAST(1);
               L Row := Row Indices'SUCC(L Row);
               R Row := Row Indices' SUCC(R Row);
            end loop Row_Loop;
      else
         --input matrices have incompatible dimensions
         raise Dimension Error;
      end if;
      return Answer;
  end "-";
```

3.3.6.2.9.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix Operations Unconstrained:

Ī	Name	Type	Description			
	Elements	floating point type	Used to define type of elements in matrix defined by this package	1		
j	Col_Indices	discrete type	Used to define second dimension of exported matrix type			
	Row_Indices	discrete type	Used to define first dimension of exported matrix type			

The following table summarizes the types required by this part and defined in the package specification for Matrix Operations_Unconstrained:

Name	Ī	Range		Description	
Matrices		N/A		Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description	
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.2.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name		When/Why Raised	- -
	Dimension_Error		Raised if the respective lengths of the first and second dimensions of the input matrices are not the same	



3.3.6.2.9.2.10.3 "+" (MATRICES + ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P350-0)

This function calculates a scaled matrix by adding a scale factor to each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.2.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R075.

3.3.6.2.9.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be scaled
Addend	Elements	In	Scale factor

3.3.6.2.9.2.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name			Туре		Description		
Ī	Answer	1	Matrices		Scaled	matrix	1

3.3.6.2.9.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

function "+" (Matrix : Matrices;

Addend : Elements) return Matrices is



-- -----------

3.3.6.2.9.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix Operations Unconstrained:

-	Name	Туре	Description
Ī	Elements	floating point type	Used to define type of elements in matrix defined by this package
İ	Col_Indices	discrete type	Used to define second dimension of exported matrix type
İ	Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix Operations Unconstrained:

Ī	Name	1	Range		Description	Ī
	Matrices	1	N/A		Unconstrained, two-dimensional array of Elements	



3.3.6.2.9.2.10.3.8 LIMITATIONS

None.

3.3.6.2.9.2.10.4 "-" (MATRICES - ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P351-0)

This function calculates a scaled matrix by subtracting a scale factor from each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.2.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R076.

3.3.6.2.9.2.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	Type	Mode	Description
	Matrix	Matrices	In	Matrix to be scaled
	Subtrahend	Elements	In	Scale factor

3.3.6.2.9.2.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	1	Туре	1	Description	Ī
Answer		Matrices	1	Scaled matrix	1

3.3.6.2.9.2.10.4.5 PROCESS CONTROL

Not applicable.



```
3.3.6.2.9.2.10.4.6 PROCESSING
```

```
The following describes the processing performed by this part:
  function "-" (Matrix : Matrices;
               Subtrahend: Elements) return Matrices is
     _____
     --declaration section-
     _____
     Answer : Matrices(Matrix'RANGE(1), Matrix'RANGE(2));
-- --begin function "-"
-- -----
  begin
     Row Loop:
        for Row in Matrix'RANGE(1) loop
          Col Loop:
             for Col in Matrix'RANGE(2) loop
                Answer(Row, Col) := Matrix(Row, Col) - Subtrahend;
             end loop Col Loop;
        end loop Row Loop;
```

3.3.6.2.9.2.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

return Answer:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

end "-":

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

	Name	Type	Description
Ī	Elements	floating point type	Used to define type of elements in matrix defined by this package
İ	Col_Indices	discrete type	Used to define second dimension of exported matrix type
İ	Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix Operations Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.2.10.4.8 LIMITATIONS

None.

3.3.6.2.9.2.10.5 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P352-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one in which the diagonal elements equal 1.0 and all other elements equal 0.0. The input matrix must be a square matrix, but the ranges of the ndividual dimensions do not have to be the same.

3.3.6.2.9.2.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R155.

3.3.6.2.9.2.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	1	Туре	1	Mode	١	Descri	pt:	lon						
Ī	Matrix		Matrices	1		1	Matrix	to	be	made	into	an	identity	 1	

3.3.6.2.9.2.10.5.4 LOCAL DATA

Data objects:

The following data objects are maintained local to this part.

end if:

```
| Name | Type | Description
| Col_Marker | Col_Indices | Index into second dimension of matrix | Row | Row_Indices | Index into first dimension of matrix
3.3.6.2.9.2.10.5.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.2.10.5.6 PROCESSING
The following describes the processing performed by this part:
   procedure Set To Identity Matrix (Matrix: out Matrices) is
    --declaration section
      Col Marker : Col Indices;
     Row : Row Indices;
-- -- begin function Set To Identity Matrix
  begin
     -- make sure input matrix is a square matrix
     if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
         Matrix := (others => (others => 0.0));
                := Matrix'FIRST(1);
        Col Marker := Matrix'FIRST(2);
        Row Loop:
            Toop
               --set diagonal element equal to 1
               Matrix(Row, Col_Marker) := 1.0;
               exit Row Loop when Row = Matrix'LAST(1);
               Row := Row Indices'SUCC(Row);
               Col Marker := Col Indices'SUCC(Col Marker);
            end loop Row_Loop;
     else
        -- do not have a square matrix
        raise Dimension Error;
```



end Set To Identity Matrix;

3.3.6.2.9.2.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Unconstrained:

Name	Range	Description	
Matrices	N/A	Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

	Name	Ī	Description
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.2.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:



3.3.6.2.9.2.10.6 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P353-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.2.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R156.

3.3.6.2.9.2.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

111			 	Description	•
•	•		•	Matrix to be zeroed out	

3.3.6.2.9.2.10.6.4 LOCAL DATA

None.

3.3.6.2.9.2.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.6.6 PROCESSING

The following describes the processing performed by this part:

procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
begin

Matrix := (others => (others => 0.0));
end Set_To_Zero_Matrix;



3.3.6.2.9.2.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix Operations Unconstrained:

Ī	Name	Туре	Description
Ī	Elements	floating point type	Used to define type of elements in matrix defined by this package
	Col_Indices	discrete type	Used to define second dimension of exported matrix type
İ	Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Unconstrained:

Name	Range	Description	I
Matrices	N/A	Unconstrained, two-dimensional array of Elements	

3.3.6.2.9.2.10.6.8 LIMITATIONS

None.

3.3.6.2.9.2.10.7 "*" (MATRICES * MATRICES => MATRICES) UNIT DESIGN (CATALOG #P354-0)

This function multiplies an m x n matrix by an n x p matrix, returning and m x p matrix. The type of elements in each of the three matrices is the same.

The values in the result matrix are defined as:

$$a(m,p) := b(m,n) * c(n,p)$$

3.3.6.2.9.2.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.



3.3.6.2.9.2.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.2.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	1	Туре	1	Mode		 Des	 iption	 		
	Left Right		Matrices Matrices		In In					multiplicand multiplier	

3.3.6.2.9.2.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1.	Name	1	Туре	I	Value	1	Description	Ī
A	nswer		Matrices Row_Indices	•	N/A N/A		Result of multiplying two input matrices Index into rows of left and answer	-
	Left Right		Col_Indices Row Indices		N/A N/A		matrices Index into columns of left matrix Index into rows of right matrix	
P			Col_Indices		N/A		Index into rows of light matrix Index into colums of left and answer matrices	

3.3.6.2.9.2.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.2.10.7.6 PROCESSING

The following describes the processing performed by this part:

function "*" (Left : Matrices;

Right: Matrices) return Matrices is

-- --declaration section

Answer : Matrices(Left'RANGE(1), Right'RANGE(2));

M : Row_Indices;
N Left : Col_Indices;

```
133
```

```
N_Right : Rov_Indices;
           : Col Indices;
-- --begin function "*"
  begin
      --make sure dimensions are compatible
      if Left'LENGTH(2) = Right'LENGTH(1) then
        M := Left'FIRST(1);
        M_Loop:
           loop
               P := Right'FIRST(2);
               P_Loop:
                  loop
                     Answer(M,P) := 0.0;
                     N Left := Left'FIRST(2);
                     N Right := Right'FIRST(1);
                     N Loop:
                        loop
                           Answer(M,P) := Answer(M,P) +
                                          Left(M,N_Left) * Right(N_Right,P);
                           exit N Loop when N Left = Left'LAST(2);
                           N Left := Col Indices'SUCC(N Left);
                           N_Right := Row_Indices'SUCC(N_Right);
                        end loop N Loop;
                     exit P Loop when P = Right'LAST(2);
                     P := Col Indices'SUCC(P);
                  end loop P_Loop;
              exit M Loop when M = Left'LAST(1);
              M := Row_Indices'SUCC(M);
           end loop M Loop;
     else
        --dimensions are incompatible
        raise Dimension Error;
     end if;
     return Answer;
  end "*";
```

3.3.6.2.9.2.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Matrix Operations Unconstrained:

-	Name	Type	Description	Ī
	Elements	floating point type	Used to define type of elements in matrix defined by this package	
-	Col_Indices	discrete type	Used to define second dimension of exported matrix type	Ì
	Row_Indices	discrete type	Used to define first dimension of exported matrix type	

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Unconstrained:

	Name	Range	Description
	Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Ī	Name		Description
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level of Matrix_Operations_Unconstrained:



Ī	Name		Type	Description	-
	# * #		function	Operator to define the operation Elements * Elements => Elements	

3.3.6.2.9.2.10.7.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	1	When/Why	Raised	1
	Dimension_Error		Raised if not have	the inner dimensions of the input matrices de the same length	0

3.3.6.2.9.3 DYNAMICALLY SPARSE MATRIX_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P362-0)

This package defines a dynamically sparse matrix and operations on it. All elements of the matrix are stored, but most of the elements are expected to be 0. Which elements are zero does not have to remain the same. See decomposition section for the operations provided.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described at the package specification level:

Data types:

The following table describes the generic formal types required by this part:



1	Name	Туре	Description	
Ī	Elements	floating point type	Data type of elements in exported matrix type	
İ	Col_Indices	discrete type	Used to dimension exported matrix type	
j	Row_Indices	discrete type	Used to dimension exported matrix type	

3.3.6.2.9.3.4 LOCAL DATA

Data types:

The following data types were previously defined at the package specification level:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Dynamically_Sparse_Matrix_Operations_Unconstrained is
end Dynamically_Sparse_Matrix_Operations_Unconstrained;

3.3.6.2.9.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.3.8 LIMITATIONS

None.

3.3.6.2.9.3.9 LLCSC DESIGN

None.



3.3.6.2.9.3.10 UNIT DESIGN

3.3.6.2.9.3.10.1 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P363-0)

This procedure sets a square input matrix to an identity matrix. An identity matrix is one where the diagonal elements all equal 1.0, with the remaining elements equaling 0.0.

3.3.6.2.9.3.10.1.1 REQUIREMENTS ALLOCATION

See main header.

3.3.6.2.9.3.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

•	Name	١	Type	1	Mode	I	Description	1
			Matrices				Matrix being made into an identity matrix	ı

3.3.6.2.9.3.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name	ı	Туре	I	Description	Ī
	Col_Marker Row		Col_Indices Row_Indices		Index into second dimension of input matrix Index into first dimension of input matrix	

3.3.6.2.9.3.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.1.6 PROCESSING

The following describes the processing performed by this part:

procedure Set To Identity Matrix (Matrix: out Matrices) is



or more ancestral units:

Data types:

```
_____
      --declaration section
     Col Marker : Col Indices:
     Row : Row Indices;
--- --begin procedure Set_to_Identity_Matrix
begin
     --make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
        Matrix := (others => (others => 0.0));
                  := Matrix'FIRST(1);
        Col Marker := Matrix'FIRST(2);
        Row Loop:
           Ioop
              -- set diagonal element equal to 1.0
              Matrix(Row, Col_Marker) := 1.0;
              exit Row Loop when Row = Matrix'LAST(1);
                       := Row Indices'SUCC(Row);
              Col_Marker := Col_Indices'SUCC(Col_Marker);
           end loop Row Loop;
     else
        raise Dimension Error;
     end if:
  end Set to Identity Matrix;
3.3.6.2.9.3.10.1.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
The following tables describe the elements used by this part but defined in one
```

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Unconstrained:



Name	Type	Description	-
Elements	floating point type	Data type of elements in exported matrix type	-
Col_Indices	discrete type	Used to dimension exported matrix type	ĺ
Row_Indices	discrete type	Used to dimension exported matrix type	İ

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Ī	Name	I	Description	-
	limension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.3.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name		When/Why	Raised]
	Dimension_Error		Raised if	the input matrix is not a square matrix	

3.3.6.2.9.3.10.2 SET_TO_ZERO_MATRIX UNIT DESIGN (CATALOG #P364-0)

This procedure sets all elements of an input matrix to zero.

3.3.6.2.9.3.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.



3.3.6.2.9.3.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	1	Туре	I	Mode	1	Description	Ī
•	•	Matrices	•		•	Matrix to be zeroed out	Ī

3.3.6.2.9.3.10.2.4 LOCAL DATA

None.

3.3.6.2.9.3.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.3.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
begin
```

```
Matrix := (others => (others => 0.0));
end Set to Zero Matrix;
```

3.3.6.2.9.3.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Unconstrained:



Name	Type	Description	-
Elements	floating point type	Data type of elements in exported matrix type	
Col_Indices	discrete type	Used to dimension exported matrix type	ĺ
Row_Indices	discrete type	Used to dimension exported matrix type	İ

The following types are defined in the package specification for Dynamically_-Sparse Matrix Operations Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

3.3.6.2.9.3.10.2.8 LIMITATIONS

None.

3.3.6.2.9.3.10.3 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P365-0)

This function takes a square input matrix and adds it to an identity matrix by adding 1.0 to all diagonal elements of the input matrix.

3.3.6.2.9.3.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Na	ne	Type		Mode		Descri	pt	ion						
Inp	ıt	Matrices		In		Matrix	to	which	is	added	an	identity	matrix	



3.3.6.2.9.3.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Туре	Value	Description
Answer	Matrices	N/A	Result of adding an identity matrix to the input matrix
Col_Marker Row	Col_Indices Row_Indices	N/A N/A	Column index Row index

3.3.6.2.9.3.10.3.5 PROCESS CONTROL

end if;

Not applicable.

3.3.6.2.9.3.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
function Add_to_Identity (Input : Matrices) return Matrices is
```



```
exit Row Loop when Row = Input'LAST(1);
                     := Row Indices'SUCC(Row);
            Col Marker := Col Indices'SUCC(Col Marker);
         end loop Row Loop;
   else
      raise Dimension_Error;
   end if;
   return Answer;
end Add_to_Identity;
```

3.3.6.2.9.3.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Unconstrained:

Ī	Name		Туре	Description	
- 	Elements		floating point type	Data type of elements in exported matrix t	ype
İ	Col_Indices		discrete type	Used to dimension exported matrix type	İ
	Row_Indices		discrete type	Used to dimension exported matrix type	j

The following types are defined in the package specification for Dynamically -Sparse Matrix Operations Unconstrained:

Ī	Name	Ī	Range	Ī	Description	Ī
	Matrices		N/A		Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

]	Name		Description	<u> </u>
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.3.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	1	When/Why	Raised	
Ī	Dimension_Error		Raised if	the input matrix is not a square matrix	

3.3.6.2.9.3.10.4 SUBTRACT FROM IDENTITY UNIT DESIGN (CATALOG #P366-0)

This function subtracts a square input matrix from an identity matrix by negating all elements of an input matrix and then adding 1.0 to the elements on the diagonal.

3.3.6.2.9.3.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

I	Name	1	Туре	Ī	Mode	1	Description	Ī
	Input		Matrices		In		Square matrix to be subtracted from an identity matrix	

3.3.6.2.9.3.10.4.4 LOCAL DATA

Data objects:



The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting input matrix from an identity matrix
Col_Marker Row	Col_Indices Row_Indices	N/A N/A	Column index Row index

3.3.6.2.9.3.10.4.5 PROCESS CONTROL

Not applicable.

```
3.3.6.2.9.3.10.4.6 PROCESSING
```

The following describes the processing performed by this part:

```
function Subtract from Identity (Input: Matrices) return Matrices is
```

```
-- --declaration section
```

```
Answer : Matrices(Input'RANGE(1),Input'RANGE(2));
```

Col_Marker : Col_Indices;
Row : Row_Indices;

```
-- -- begin procedure Subtract_From_Identity
```

begin

```
-- --make sure input is a square matrix if Input'LENGTH(1) = Input'LENGTH(2) then

Row := Input'FIRST(1);
Col_Marker := Input'FIRST(2);
Row_Loop:
Ioop
```

Answer(Row,Col) := 0.0; end if; end loop Col Loop;

```
if Answer(Row, Col_Marker) /= 0.0 then
   Answer(Row, Col_Marker) := Answer(Row, Col_Marker) + 1.0;
else
```

3.3.6.2.9.3.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices		Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse Matrix Operations_Unconstrained:

Ī	Name		Range		Description	Ī
	Matrices		N/A		Unconstrained, two-dimensional array of Elements	

Exceptions:



The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description	
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.3.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	<u> </u>	When/Why	Raised	
Ī	Dimension_Error	 -	Raised if	the input matrix is not a square matrix	-

3.3.6.2.9.3.10.5 "+" UNIT DESIGN (CATALOG #P367-0)

This function adds two sparse m x n matrices, by adding the individual elements of the input matrices taking advantage of the fact that most of the elements of both matrices equal 0.

3.3.6.2.9.3.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	Type	Mode	Description	Ī
	Left Right	Matrices Matrices	In In	Sparse matrix to be added Sparse matrix to be added	



3.3.6.2.9.3.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description	1
Answer L Col I. Row R Col R Row	Col_Indices Row_Indices Col_Indices	N/A N/A N/A N/A N/A	Result of adding two input matrices Column index into left matrix Row index into left matrix Column index into right matrix Row index into right matrix	

3.3.6.2.9.3.10.5.5 PROCESS CONTROL

function "+" (Left : Matrices;

Not applicable.

3.3.6.2.9.3.10.5.6 PROCESSING

The following describes the processing performed by this part:

Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
L_Col : Col_Indices;
L_Row : Row_Indices;
R_Col : Col_Indices;
R_Row : Row_Indices;

-- --begin function "+"

begin

```
-- -- make sure have compatible dimensions
if Left'LENGTH(1) = Right'LENGTH(1) and then
Left'LENGTH(2) = Right'LENGTH(2) then

L_Row := Left'FIRST(1);
R_Row := Right'FIRST(1);
Row Loop:
Toop

L_Col := Left'FIRST(2);
R_Col := Right'FIRST(2);
Col Loop:
```

loop

MALL (O)

```
if Left(L_Row, L_Col) = 0.0 then
                          if Right(R Row, R Col) = 0.0 then
                             Answer(\overline{L} Row, \overline{L} Col) := 0.0;
                             Answer(L Row, L Col) := Right(R Row, R Col);
                         end if:
                      elsif Right(R Row, R Col) = 0.0 then
                         Answer(L Row, L Col) := Left(L Row, L Col);
                      else
                         Answer(L Row, L Col) := Left(L Row, L Col) +
                                                   Right(\overline{R}_Row, \overline{R}_Col);
                      end if;
                      exit Col Loop when L Col = Left'LAST(2);
                      L Col := Col Indices'SUCC(L Col);
                      R Col := Col Indices'SUCC(R Col);
                   end loop Col Loop;
                exit Row Loop when L Row = Left'LAST(1);
                L Row := Row Indices SUCC(L Row);
                R_Row := Row_Indices'SUCC(R_Row);
             end loop Row_Loop;
      else
          raise Dimension Error;
      end if;
      return Answer;
   end "+";
3.3.6.2.9.3.10.5.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
The following tables describe the elements used by this part but defined in one
or more ancestral units:
Data types:
The following table summarizes the generic formal types visible to this part
and defined in the package specification for Dynamically Sparse Matrix -
Operations_Unconstrained:
```



Name	Type	Description	-
Elements	floating point type	Data type of elements in exported matrix type	
Col_Indices	discrete type	Used to dimension exported matrix type	İ
Row_Indices	discrete type	Used to dimension exported matrix type	

The following types are defined in the package specification for Dynamically_-Sparse Matrix Operations Unconstrained:

Name	Range	Description	
Matrice	es N/A	Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	I	Description	
dimension_er	ror	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.3.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name		When/Why	Raised	1
Dimensi	on_Error	Raised if	both matrices are not m x n matrices]

3.3.6.2.9.3.10.6 "-" UNIT DESIGN (CATALOG #P368-0)

This function subtracts two sparse $m \times n$ matrices by subtracting the individual elements of the input matrices, taking advantage of the fact that most of the elements of both matrices equal 0.



3 3.6.2.9.3.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.3.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.3.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Туре	1	Mode	Ī	Description	-
	Left Right		Matrices Matrices		In In		Sparse matrix to be treated as the minuend Sparse matrix to be treated as the subtrahend	

3.3.6.2.9.3.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name		Туре	I	Value		Description	Ī
	Answer L_Col L_Row R_Col R_Row	İ	Matrices Col_Indices Row_Indices Col_Indices Row_Indices	İ	N/A N/A N/A N/A N/A		Result of subtracting two input matrices Column index into left matrix Row index into left matrix Column index into right matrix Row index into right matrix	

3.3.6.2.9.3.10.6.5 PROCESS CONTROL

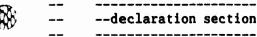
Not applicable.

3.3.6.2.9.3.10.6.6 PROCESSING

The following describes the processing performed by this part:

function "-" (Left : Matrices;

Right: Matrices) return Matrices is





```
Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
      L_Col : Col_Indices;
      L Row : Row Indices;
      R Col : Col Indices;
      R Row : Row Indices;
-- --begin function "-"
   begin
      --make sure have compatible dimensions
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then
         L Row := Left'FIRST(1);
         R Row := Right'FIRST(1);
         Row Loop:
            Toop
                L Col := Left'FIRST(2);
                R Col := Right'FIRST(2);
                Col Loop:
                   Ioop
                      if Left(L_Row, L_Col) = 0.0 then
                         if Right(R Row, R Col) = 0.0 then
                             Answer(\overline{L}_{Row}, \overline{L}_{Col}) := 0.0;
                             Answer(L_Row, L_Col) := - Right(R_Row, R_Col);
                         end if;
                      elsif Right(R Row, R Col) = 0.0 then
                         Answer(L_Row, L_Col) := Left(L_Row, L_Col);
                      else
                         Answer(L_Row, L_Col) := Left(L_Row, L_Col) -
                                                   Right(\overline{R} Row, \overline{R} Col);
                      end if;
                      exit Col Loop when L Col = Left'LAST(2);
                      L Col := Col Indices SUCC(L Col);
                      R Col := Col Indices'SUCC(R Col);
                   end loop Col_Loop;
                exit Row Loop when L Row = Left'LAST(1);
                L Row := Row Indices SUCC(L Row);
               R Row := Row Indices' SUCC(R Row);
            end loop Row Loop;
      else
         raise Dimension Error;
      end if;
```



return Answer;

end "-";

3.3.6.2.9.3.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Unconstrained:

1	Name	T	 уре	Desc	ription
Ī	Elements		oating oint type		type of elements in exported matrix type
İ	Col_Indices	di	screte ype		to dimension exported matrix type
	Row_Indices	di	screte ype	Used	to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse Matrix Operations Unconstrained:

Name	Range	Description
Matrices	N/A	Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description	1
dimension_er	ror Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	1



3.3.6.2.9.3.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	1	When/Why	Raised	
Dimension_Error		Raised if	both matrices are not m x n matrices	•

3.3.6.2.9.4 SYMMETRIC_HALF_STORAGE_MATRIX_OPERATIONS PACKAGE DESIGN (CATALOG #P376-0)

This package defines a symmetric half storage matrix and provides operations on it. For the operations provided, see the decomposition section. The bottom half of the matrix will be stored in row-major order.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.2 LOCAL ENTITIES DESIGN

Subprograms:

The following table describes the subprograms local to this part:

Ī	Name	Туре	Description	-
1	Swap_Row	function	Takes a column as input, and returns the corresponding row entry	Ī
	Swap_Col	function	Takes a row as input, and returns the corresponding column entry	

This package contains code which is executed when the package is elaborated. This code first checks to make sure a square matrix has been instantiated. If not, a Dimension Error exception is raised. If a square matrix has been instantiated, this the code initializes the Row_Marker, Local_Identity_Matrix, and Col_Offset arrays.

3.3.6.2.9.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification.



Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col Indices	discrete type	Used to dimension column slices
Row Indices	discrete type 	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored
Col Slices	array	Data type defining a column slice of a matrix
Row_ Slices	array	Data type defining a row slice of a matrix

3.3.6.2.9.4.4 LOCAL DATA

Data types:

The following table describes the data types previously defined in this part's package specification:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

The following table describes the data types defined local to this package.

Name	Type	Description
Col_Index_Arrays	array	Array of integers indexed by Indices used to set up column markers into Matrices array
Row_Index_Arrays	array	Array of integers indexed by Indices used to set up row markers into Matrices array

Data objects:

The following table describes the data objects defined in this part's package specification:



-	Name		Туре		Value	 	Description	-
	Entry_Count		Positive				Number of stored values from the half- storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2	

The following table describes the data objects maintained by this package:

Name	Type	Description
Col_Offset	Col_Index_ Arrays	Used to determine the offset of a column index from the first column index
Row_Marker	Row_Index_ Arrays	Used to marker where in Matrices a particular row begins
Local_ Identity_ Matrix	Matrices 	Pre-initialized identity matrix
Local Zero Matrīx	Matrices	Pre-initialized zero matrix

Note: The following scheme is used to access an element:

Full_Storage(i,j) <==>
Half_Storage(Row_Marker(i) + Col_Offset(j));

The following data objects are contained in a declare block located at the end of this package body:

Name	Type Value	Description
Count	Natural N/A	Counts the position of the row index; goes from 0 to the number of rows - 1
0ffset	Natural N/A	Offset of the current column index from the first column index
Row_Starting_Point	Natural N/A	Where in the diagonal matrix the current row starts

3.3.6.2.9.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.6 PROCESSING

The following describes the processing performed by this part: separate (General_Vector_Matrix_Algebra)

```
CAMP Software Detailed Design Document
package body Symmetric Half Storage Matrix Operations is
__ ____
-- --local declarations
   type Col_Index_Arrays is array(Col_Indices) of NATURAL;
   type Row Index Arrays is array(Row Indices) of NATURAL;
  Col Offset : Col Index Arrays;
  Row_Marker : Row_Index_Arrays;
-- -- this object is initially only zeroed out; the 1.0 values will be assigned
-- -- to the diagonal elements during package initialization
  Local Identity Matrix : Matrices := (others => 0.0);
  Local_Zero_Matrix : constant Matrices := (others => 0.0);
--begin processing for Symmetric Half Storage
--Matrix Operations package body
begin
  Init Block:
     declare
                 : NATURAL;
        Count
                         : NATURAL;
        Row_Starting_Point : NATURAL;
     begin
        --make sure lengths of row and col indices are the same
        if Row_Slices'LENGTH /= Col_Slices'LENGTH then
           raise Dimension Error;
        else
          --initialize row marker identity matrix arrays;
          --all diagonal elements, except for the last one, which require
          -- a value of 1 for the identity matrix are located one entry
          -- before the starting location of the next row
          --handle first row marker entry to simplify initialization of
          -- the identity matrix -- (NOTE: count implicitly equals 0)
          Row Marker(Row Indices'FIRST) := 1;
          Count := 1;
```

Row Marker and Identity Matrix Init Loop:

for Index in Row Indices'SUCC(Row Indices'FIRST) .. Row Indices'LAST loop



```
Row Starting Point := (Count * (Count+1) / 2) + 1;
                 Row Marker(Index) := Row Starting Point;
                 Local Identity Matrix(Row Starting Point-1) := 1.0;
                 Count := Count + 1;
              end loop Row Marker and Identity Matrix Init Loop;
           --initialize last diagonal element
           Local_Identity_Matrix(Entry_Count) := 1.0;
           --initialize column offset array
           _____
           Offset := 0;
           Col Marker Init Loop:
              for Index in Col Indices loop
                 Col Offset(Index) := Offset;
                 Offset := Offset + 1;
              end loop Col Marker Init Loop;
        end if;
     end Init Block;
end Symmetric_Half_Storage_Matrix_Operations;
```

3.3.6.2.9.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Ī	Name		Description	Ī
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:



-	Name	1	When/Why	 7	Raised							
	Dimension_Error		Raised i	E	an attempt matrix	is	made	to	instantiated	other	than	

3.3.6.2.9.4.9 LLCSC DESIGN

None.

3.3.6.2.9.4.10 UNIT DESIGN

3.3.6.2.9.4.10.1 SWAP_COL UNIT DESIGN

This function takes a row index is input, and returns the corresponding column index.

3.3.6.2.9.4.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type		Mode	•	Description	Ī
Row	Row_Indices		In		Row to be converted to a column entry	Ī

3.3.6.2.9.4.10.1.4 LOCAL DATA

None.

3.3.6.2.9.4.10.1.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.4.10.1.6 PROCESSING

The following describes the processing performed by this part:

3.3.6.2.9.4.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Ī	Name	Type	Description
	Col !ndices Row Indices	discrete type discrete type	Used to dimension column slices Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

3.3.6.2.9.4.10.1.8 LIMITATIONS

None.

3.3.6.2.9.4.10.2 SWAP ROW UNIT DESIGN

This function takes a column index is input, and returns the corresponding row index.

3.3.6.2.9.4.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.2.2 LOCAL ENTITIES DESIGN

None.



3.3.6.2.9.4.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type		Mode	1	Descri	pti	on						
Col	Col_Indices	1	In		Column	to	be	converted	to	a	row	entry	

3.3.6.2.9.4.10 2.4 LOCAL DATA

None.

3.3.6.2.9.4.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.2.6 PROCESSING

The following describes the processing performed by this part:

3.3.6.2.9.4.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name Type	Description
Col	Used to dimension column slices Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored



3.3.6.2.9.4.10.2.8 LIMITATIONS

None.

3.3.6.2.9.4.10.3 INITIALIZE UNIT DESIGN (CATALOG #P379-0)

This function allows the half storage matrix to be initialized a row at a time. Even through an entire row of values is sent in, only those applicable to a given row will be referenced (i.e., 1 value for row 1, 2 values for row 2, etc.).

3.3.6.2.9.4.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode	1	Description	Ī
	Row_Slice	Slices	In		Row of values to be placed in the half- storage matrix	
	Row	Indices	In		Indicates which row of values this is	Ì
İ	Matrix	Matrices	Out	İ	Half-storage matrix to be initialized	İ

3.3.6.2.9.4.10.3.4 LOCAL DATA

None.

3.3.6.2.9.4.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.3.6 PROCESSING

The following describes the processing performed by this part:

Matrix : out Matrices) is

3.3.6.2.9.4.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Туре	Description
Elements	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col Indices	discrete type	Used to dimension column slices
Row Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored
Col Slīces	array	Data type defining a column slice of a matrix
Row Slices	array	Data type defining a row slice of a matrix

The following table summarizes the types required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

1	Name	Range	Description
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Туре	Description
Col_Offset Row_Marker	Col_Index_ Arrays Row_Index_ Arrays	Used to determine the offset of a column index from the first column index Used to marker where in Matrices a particular row begins

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

]	Name	Type	Description	
	Swap_Row	function	Takes a column as input, and returns the corresponding row entry Takes a row as input, and returns the corresponding	
1	Swap_cor	lunction	column entry	



3.3.6.2.9.4.10.3.8 LIMITATIONS

None.

3.3.6.2.9.4.10.4 IDENTITY MATRIX UNIT DESIGN (CATALOG #P380-0)

This function returns an identity matrix. An identity matrix is one where all elements equal 0.0 except the diagonal elements which equal 1.0.

3.3.6.2.9.4.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.4.3 INPUT/OUTPUT

None.

3.3.6.2.9.4.10.4.4 LOCAL DATA

None.

3.3.6.2.9.4.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.4.6 PROCESSING

The following describes the processing performed by this part:

function Identity Matrix return Matrices is

begin

return Local Identity Matrix;

end Identity_Matrix;

3.3.6.2.9.4.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:



Data types:

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

1	Name	Range	Description
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Type	Description	
Local_ Identity_ Matrix	Matrices	Pre-initialized identity matrix	

3.3.6.2.9.4.10.4.8 LIMITATIONS

None.

3.3.6.2.9.4.10.5 ZERO MATRIX UNIT DESIGN (CATALOG #P381-0)

This function returns a zeroed out half-storage matrix.

3.3.6.2.9.4.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.5.3 INPUT/OUTPUT

None.



3.3.6.2.9.4.10.5.4 LOCAL DATA

None.

3.3.6.2.9.4.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.5.6 PROCESSING

The following describes the processing performed by this part:

function Zero_Matrix return Matrices is

begin

return Local_Zero_Matrix;

end Zero_Matrix;

3.3.6.2.9.4.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the types required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

Ī	Name	Range	Description
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric Half Storage Matrix_Operations:

Name	Type	Description
Local_ Zero Matrix	Matrices	Pre-initialized zero matrix



3.3.6.2.9.4.10.5.8 LIMITATIONS

None.

3.3.6.2.9.4.10.6 CHANGE ELEMENT UNIT DESIGN (CATALOG #P382-0)

This procedure changes a single element in the half-storage matrix based on the two-dimensional row and column indices provided.

3.3.6.2.9.4.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

-	Name	Type Mode Description	
	New_Value Row Col Matrix	lements In	d

3.3.6.2.9.4.10.6.4 LOCAL DATA

None.

3.3.6.2.9.4.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.6.6 PROCESSING

The following describes the processing performed by this part:

begin



3.3.6.2.9.4.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Тур €	Description
Elements Col Indices Row Indices	floating point type discrete type discrete type	Data type of elements in the half storage matrix and in the Slices array Used to dimension column slices Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

I	Name	Range	Description
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order



Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric Half Storage Matrix Operations:

Name	Type	Description	
i -	Col_Index_ Arrays	Used to determine the offset of a column index from the first column index	
Row_Marker	Row_Index_ Arrays	Used to marker where in Matrices a particular row begins	

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

]	Name	Type	Description	- -
	Swap_Row	function	Takes a column as input, and returns the corresponding row entry	
	Swap_Col	function	Takes a row as input, and returns the corresponding column entry	İ

3.3.6.2.9.4.10.6.8 LIMITATIONS

None.

3.3.6.2.9.4.10.7 RETRIEVE ELEMENT UNIT DESIGN (CATALOG #P383-0)

This function retrieves an element from a half-storage matrix using two-dimensional row and column indices as input.

3.3.6.2.9.4.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:



The following table describes this part's formal parameters:

Name	Type	Mode	Description	1
Matrix	Matrices	In	Half-storage matrix containing desired	
Row	Row_Indices	In	Row in which element is contained	i

3.3.6.2.9.4.10.7.4 LOCAL DATA

None.

3.3.6.2.9.4.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.7.6 PROCESSING

The following describes the processing performed by this part:

function Retrieve Element (Matrix : Matrices;

Row : Row Indices;

Col : Col Indices) return Elements is

-- --declaration section

Answer : Elements:

-- --begin function Retrieve_Element

begin

--determine which half of the array is being referenced if Row Indices'POS(Row) - Row Indices'POS(Row Indices'FIRST) >= Col_Indices'POS(Col) - Col_Indices'POS(Col_Indices'FIRST) then

-- -- already looking at the bottom half of the array
Answer := Matrix(Row_Marker(Row) + Col_Offset(Col));

else

end if;

return Answer;

end Retrieve_Element;

3.3.6.2.9.4.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Туре	Description
Col Indices Row Indices	floating point type discrete type discrete type	Data type of elements in the half storage matrix and in the Slices array Used to dimension column slices Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

1	Name	Ī	Туре	ı	Description	Ī
İ	Col_Offset	İ	Arrays -	İ	Used to determine the offset of a column index from the first column index	
Í	Row_Marker		Row_Index_ Arrays	İ	Used to marker where in Matrices a particular row begins	



Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric Half Storage Matrix -Operations:

Ī	Name	 	Type	ļ	Description	1
ļ	Swap_Row		function		Takes a column as input, and returns the corresponding row entry	
į	Swap_Col	İ	function	İ	Takes a row as input, and returns the corresponding column entry	İ

3.3.6.2.9.4.10.7.8 LIMITATIONS

None.

3.3.6.2.9.4.10.8 ROW SLICE UNIT DESIGN (CATALOG #P384-0)

This function returns an array which contains all the elements in a requested row.

3.3.6.2.9.4.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.8.3 INPUT/OUTPUT

FORMAL FARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	1	Туре	1	Mode	İ	Description	
Ī	Matrix		Matrices		In		Half-storage matrix containing values to be retrieved	be
	Row	İ	Row_Indices		In		Indicates which row of values is desired	İ

3.3.6.2.9.4.10.8.4 LOCAL DATA

Data objects:



The following table describes the data objects maintained by this part:

```
| Name | Type | Value | Description
______
| Answer | Row Slices | N/A | Requested row of values
3.3.6.2.9.4.10.8.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.4.10.8.6 PROCESSING
The following describes the processing performed by this part:
  function Row Slice (Matrix : Matrices;
                       : Row Indices) return Row Slices is
                   Row
    --declaration section
     Answer : Row Slices;
-- -- begin function Row Slice
  begin
     --retrieve row elements in bottom half of array
     Bottom Loop:
       for Col in Col Indices'FIRST .. Swap Col(Row) loop
          Answer(Col) := Matrix(Row_Marker(Row) + Col_Offset(Col));
       end loop Bottom Loop;
     --retrieve row elements in top half of array, if there are any
     if Row /= Row Indices'LAST then
       Top_Loop:
          For Col in Col Indices'SUCC(Swap Col(Row)) .. Col Indices'LAST loop
            Answer(Col) := Matrix(Row Marker(Swap Row(Col)) +
                               Col Offset(Swap Col(Row)));
          end loop Top Loop;
     end if;
     return Answer
  end Row_Slice;
```

3.3.6.2.9.4.10.8.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by the .rt but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name	Туре	Description
Col Indices Row	discrete type discrete	Used to dimension column slices
Indices	type	matrix and to determine the number of elements which need to be stored
Row Slices	array 	Data type defining a row slice of a matrix

The following table summarizes the types required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

Ī	Name	Range	Description	1
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order	

Data objects:

The following data objects are required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Ī	Name	1	Туре	Ī	Description	1
	Col_Offset				Used to determine the offset of a column index from the first column index	
İ	Row_Marker		Row_Index_ Arrays		Used to marker where in Matrices a particular row begins	

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:





-	Name	Туре	Description	1
 	Swap_Row	function	Takes a column as input, and returns the corresponding row entry	-
j	Swap_Col	function	Takes a row as input, and returns the corresponding column entry	İ

3.3.6.2.9.4.10.8.8 LIMITATIONS

None.

3.3.6.2.9.4.10.9 COLUMN SLICE UNIT DESIGN (CATALOG #P385-0)

This function retrieves all the values contained in a single column of a symmetric matrix.

3.3.6.2.9.4.10.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.9.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Туре	Ī	Mode	I	Description	
-	Matrix		Matrices		In		Half-storage matrix from which a column of values is to be retrieved	
İ	Col	İ	Col_Indices		In	İ	Indicates which column of values is desired	ĺ

3.3.6.2.9.4.10.9.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name	•		•	•	Description	
			Col_Slices		 	Column's worth of values	

Operations:

```
3.3.6.2.9.4.10.9.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.4.10.9.6 PROCESSING
The following describes the processing performed by this part:
   function Column_Slice (Matrix : Matrices;
                          Col : Col Indices) return Col Slices is
      --declaration section
      Answer : Col Slices;
-- --begin function Column Slice
   begin
      -- retrieve column elements contained in bottom half of array
      Bottom Loop:
         for Row in Swap Row(Col) .. Row Indices'LAST loop
            Answer(Row) := Matrix(Row Marker(Row) + Col Offset(Col));
         end loop Bottom Loop;
      --retrieve column elements contained in top half of array, if any
      if Col /= Col Indices'FIRST then
         Top Loop:
            For Row in Row Indices'FIRST .. Row Indices'PRED(Swap Row(Col)) loop
               Answer(Row) = Matrix(Row Marker(Swap Row(Col)) +
                                     Col Offset(Swap Col(Row)));
            end loop Top Loop;
      end if;
      return Answer;
   end Column Slice;
3.3.6.2.9.4.10.9.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
The following tables describe the elements used by this part but defined in one
or more ancestral units:
Data types:
The following table summarizes the generic types available to this part and
defined at the package specification level of Symmetric Half_Storage_Matrix -
```

Name	Type	Description
Elements Col Indices	floating point type discrete type	Data type of elements in the half storage matrix and in the Slices array Used to dimension column slices
Row Indices Col Slices	discrete type array	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored Data type defining a column slice of a matrix

The following table summarizes the types required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

1	Name	Range	Description
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

	Name		Туре	 	Description	Ī
İ	Col_Offset Row Marker	İ	Col_Index_ Arrays Row Index	j	Used to determine the offset of a column index from the first column index Used to marker where in Matrices a particular	
	WOW THAT WEL		Arrays		row begins	

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Ī	Name		Туре	Ī	Description	Ī
		İ		İ	Takes a column as input, and returns the corresponding row entry Takes a row as input, and returns the corresponding	-
	Swap_Col		tunction	İ	column entry	



3.3.6.2.9.4.10.9.8 LIMITATIONS

None.

3.3.6.2.9.4.10.10 ADD TO IDENTITY UNIT DESIGN (CATALOG #P386-0)

This function adds an input matrix to an identity matrix, returning the result. The addition is performed by adding 1.0 to each diagonal element of the input matrix.

3.3.6.2.9.4.10.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.10.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type Mode	Description	1
Input	Matrices In	Matrix to be added to an identity matrix	1

3.3.6.2.9.4.10.10.4 LOCAL DATA

Data objects:

The following table summarizes the data objects maintained by this part:

•	Name	1	Туре		Description	
					Result of adding input matrix to identity matrix	

3.3.6.2.9.4.10.10.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.4.10.10.6 PROCESSING The following describes the processing performed by this part: function Add_to_Identity (Input : Matrices) return Matrices is --declaration section _____ Answer : Matrices; -- --begin function Add To Identity begin --do straight assignment of all elements and then add in the --identity matrix Answer := Input; --all diagonal elements, except for the last one, are located one --entry before the starting location of the next row Add Identity Loop: for Index in Row Indices' SUCC(Row Indices' FIRST) .. Row Indices' LAST loop Answer(Row Marker(Index) - 1) := Answer(Row Marker(Index)-1) + 1.0; end loop Add Identity Loop; --handle last diagonal element Answer(Entry_Count) := Answer(Entry_Count) + 1.0; return Answer; end Add to Identity;

3.3.6.2.9.4.10.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:



Name Type	Description	
Elements floating point type Row discrete Indices type	Data type of elements in the half storage matrix and in the Slices array Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored	

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

1	Name	Range	Description
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

1	Name		Туре		Value		Description
	Entry_Count		Positive				Number of stored values from the half- storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Name	Туре	Description
Col_Offset	Col_Index_ Arrays	Used to determine the offset of a column index from the first column index Used to marker where in Matrices a particular
Row_Marker	Row_Index_ Arrays	Used to marker where in Matrices a particular row begins

3.3.6.2.9.4.10.10.8 LIMITATIONS

None.



3.3.6.2.9.4.10.11 SUBTRACT FROM IDENTITY UNIT DESIGN (CATALOG #P387-0)

This function subtracts an input matrix from an identity matrix. It does this by first subtracting the input matrix from a zero matrix and then adding it to an identity matrix.

3.3.6.2.9.4.10.11.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.11.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.11.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	Type	Mode	Description	
	Input	Matrices	In	Matrix to be subtracted from an identity matrix	

3.3.6.2.9.4.10.11.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description	
Answer	Matrices	Result of subtracting input matrix from an identity matrix	

3.3.6.2.9.4.10.11.5 PROCESS CONTROL

Not applicable.

3.3 1.2.9.4.10.11.6 PROCESSING

The following describes the processing performed by this part:

function Subtract_from_Identity (Input : Matrices) return Matrices is

```
--declaration section
      Answer : Matrices:
-- -- begin function Subtract from Identity
   begin
      --subtract Input from a zero matrix and then add it to an identity matrix
      Subtract Loop:
         for Index in 1..Entry_Count loop
            Answer(Index) := - Input(Index);
         end loop Subtract Loop;
      --all diagonal elements, except for the last one, are located one
      --entry before the starting location of the next row
      Add Identity Loop:
         for Index in Row Indices' SUCC(Row Indices' FIRST) ..
                      Row Indices' LAST loop
            Answer(Row Marker(Index) - 1) := Answer(Row Marker(Index)-1) + 1.0;
         end loop Add Identity Loop;
      --handle last diagonal element
      Answer(Entry Count) := Answer(Entry Count) + 1.0;
      return Answer:
   end Subtract from Identity;
3.3.6.2.9.4.10.11.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
The following tables describe the elements used by this part but defined in one
or more ancestral units:
DAF THOS!
The land and in the summarizes the generic types available to this part and
deil____ the package specification level of Symmetric_Half_Storage_Matrix_-
Operat__as:
```



Name	Туре	Description
j	floating point type	Data type of elements in the half storage matrix and in the Slices array
Col_ Indices	discrete type	Used to dimension column slices
Row Indices	discrete type	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored

The following table summarizes the types required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

Name	Range	Description
Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

1	Name	1	Туре	1	Value		Description
	Entry_Count		Positive				Number of stored values from the half- storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2

The following table summarizes the objects required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_Operations:

Ī	Name	I	Туре		Description	Ī
	Col_Offset		Col_Index_ Arrays		Used to determine the offset of a column index from the first column index	
	Row_Marker	İ	Row_Index_ Arrays		Used to marker where in Matrices a particular row begins	İ

3.3.6.2.9.4.10.11.8 LIMITATIONS

None.

3.3.6.2.9.4.10.12 "+" UNIT DESIGN (CATALOG #P388-0)

This function adds two half-storage matrices by adding the individual elements of the input matrices, returning the resultant matrix.

3.3.6.2.9.4.10.12.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.12.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.12.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	•	Description
Left	Matrices	In	First matrix to be added Second matrix to be added
Right	Matrices	In	

3.3.6.2.9.4.10.12.4 LOCAL DATA

Data objects:

The following data objects are maintained by this part:

1	Name	1	Туре	I	Descripti	on					
I	Answer	1	Matrices	1	Result of	adding	the	tvo	input	matrices	

3.3.6.2.9.4.10.12.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.12.6 PROCESSING

The following describes the processing performed by this part:

function "+" (Left : Matrices;

Right: Matrices) return Matrices is

-- --declaration section

3.3.6.2.9.4.10.12.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Name Type	Description
Elements floating point type discrete Indices type Row discrete Indices type	Data type of elements in the half storage matrix and in the Slices array Used to dimension column slices Used to dimension row slices of the half storage matrix and to determine the number of elements

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

Name	Range	Description
Matrices	N/A 	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order



Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

Ī	Name	1	Туре	1	Value	1	Description	1
	Entry_Count		Positive				Number of stored values from the half- storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2	

The following table summarizes the objects required by this part and defined in the package body of Symmetric Half Storage Matrix Operations:

- 	Name	Type	Description	-
	Col_Offset	Col_Index_ Arrays	Used to determine the offset of a column index from the first column index	
İ	Row_Marker	Row_Index_ Arrays	Used to marker where in Matrices a particular row begins	İ

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Ī	Name	 -	Туре		Description	
	Swap_Row]]	function		Takes a column as input, and returns the corresponding row entry	
İ	Swap_Col	İ	function	İ	Takes a row as input, and returns the corresponding column entry	

3.3.6.2.9.4.10.12.8 LIMITATIONS

None.

3.3.6.2.9.4.10.13 "-" UNIT DESIGN (CATALOG #P389-0)

This function subtracts two half-storage matrices by subtracting the individual elements of the input matrices, returning the resultant matrix.



3.3.6.2.9.4.10.13.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R211.

3.3.6.2.9.4.10.13.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.4.10.13.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Туре	1	Mode		Description	<u>-</u>
Ī	Left Right		Matrices Matrices		In In		Matrix to be subtracted from Matrix to be subtracted from Left	Ī

3.3.6.2.9.4.10.13.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name		Туре		Value		Description	Ī
	Answer		Matrices		N/A		Result of subtracting Right input matrix from Left input matrix	

3.3.6.2.9.4.10.13.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.4.10.13.6 PROCESSING

The following describes the processing performed by this part:

function "-" (Left : Matrices;

Right: Matrices) return Matrices is

-- --declaration section

Answer : Matrices;

__ ____

```
33
```

```
-- --begin function "-"

begin

Process:
    for Index in 1 .. Entry Count loop
        Answer(Index) := Left(Index) - Right(Index);
    end loop Process;

return Answer;
end "-";
```

3.3.6.2.9.4.10.13.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types available to this part and defined at the package specification level of Symmetric_Half_Storage_Matrix_-Operations:

Ī	Name	Туре	Description	Ī
	Elements	point type	Data type of elements in the half storage matrix and in the Slices array	-
	Col Indices	discrete type	Used to dimension column slices	
	Row Indices	discrete type 	Used to dimension row slices of the half storage matrix and to determine the number of elements which need to be stored	

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Half_Storage_Matrix_Operations:

-	Name	Range	Description
	Matrices	N/A	A one-dimensional representation of a two-dimensional, half-storage matrix; the bottom half of the matrix will be stored in row-major order

Data objects:



The following table summarizes the objects required by this part and defined in the package specification of Symmetric Half Storage Matrix Operations:

Name	: :	1	Туре	1	Value		Description	
Entry	_Count		Positive				Number of stored values from the half- storage matrix; the number of elements stored in a half-storage matrix with n rows elements is n(n+1)/2	

The following table summarizes the objects required by this part and defined in the package body of Symmetric Half Storage Matrix Operations:

Ī	Name		Туре		Description	
Ī	Col_Offset		Col_Index_ Arrays		Used to determine the offset of a column index from the first column index	
İ	Row_Marker		Row_Index_ Arrays	İ	Used to marker where in Matrices a particular row begins	İ

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Symmetric_Half_Storage_Matrix_-Operations:

Ī	Name	Type	Description
Ī	Swap_Row	function	Takes a column as input, and returns the corresponding row entry
	Swap_Col	function	Takes a row as input, and returns the corresponding column entry

3.3.6.2.9.4.10.13.8 LIMITATIONS

None.

3.3.6.2.9.5 SYMMETRIC FULL_STORAGE_MATRIX_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P390-0)

This package exports operations on a symmetric full storage matrix.

The decomposition for this part is the same as that shown in the Top-Level Design Document.



3.3.6.2.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Name	Type	Description	
Elements	floating point type	Data type of elements in exported matrix type	
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type	j

3.3.6.2.9.5.4 LOCAL DATA

Exceptions:

The following table describes the exceptions defined in this part's package specification:

Name		Description	
Invalid_Index		Indicates an attempt was made to access an element beyond the dimensions of the array	

Data types:

The following types are previously defined in this part's package specification:

1	Name	Range	Description	I
	Matrices	N/A	Unconstrained, two-dimensional array of Elements	



3.3.6.2.9.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Symmetric Full Storage Matrix Operations Unconstrained is

end Symmetric_Full_Storage_Matrix_Operations_Unconstrained;

3.3.6.2.9.5.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.5.8 LIMITATIONS

None.

3.3.6.2.9.5.9 LLCSC DESIGN

None.

3.3.6.2.9.5.10 UNIT DESIGN

3.3.6.2.9.5.10.1 CHANGE ELEMENT UNIT DESIGN (CATALOG #P391-0)

This procedure changes the indicated element of a symmetric matrix, along with its symmetric counterpart.

3.3.6.2.9.5.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

•	٠.	
`	1.	
١,	7	

Name	Type	Mode	Description	-
New_Value Row Col Matrix	Elements Row_Indices Col_Indices Matrices	In In In In/Out	New value to be placed in the matrix Row in which the value belongs Column in which the value belongs Matrix being updated	

3.3.6.2.9.5.10.1.4 LOCAL DATA

None.

3.3.6.2.9.5.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.1.6 PROCESSING

The following describes the processing performed by this part:

procedure Change_Element (New_Value : in Elements; Row : in Row Indices; Col : in Col Indices; Matrix : in out Matrices) is

--declaration section-

S Col : Col Indices; S_Row : Row_Indices;

-- --begin procedure Change_Element-

begin

--make sure you have a square matrix if Matrix'LENGTH(1) /= Matrix'LENGTH(2) then

raise Dimension Error;

-- make sure row and col are within bounds elsif NOT (Row in Matrix'RANGE(1) and Col in Matrix'RANGE(2)) then

raise Invalid Index;

else

--everything is okay

3.3.6.2.9.5.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Unconstrained:

Ī	Name	Ī	Туре	Ī	Description	1
Ī	Elements		floating point type		Data type of elements in exported matrix type	
	Col_Indices Row_Indices		discrete type discrete type		Used to dimension exported matrix type Used to dimension exported matrix type	

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Name	Range	Description	
Matrices	N/A 	Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package body of Symmetric Full Storage Matrix Operations Unconstrained:



Ī	Name	 	Description	-
	Invalid_Index		Indicates an attempt was made to access an element beyond the dimensions of the array	

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name		Description	
Dimension_Error		Raised by a routine or package when input received has dimensions incompatible with the type of operation to be performed	

3.3.6.2.9.5.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised
Invalid_Index	Raised if an attempt is made to place an element outside the bounds of the input array
Dimension_ Error	Raised if the input matrix is not a square matrix

3.3.6.2.9.5.10.2 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P392-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one where all elements equal 0.0, except those on the diagonal which equal 1.0. The input matrix must be a square matrix, but the ranges of the individual dimensions do not have to be the same.

3.3.6.2.9.5.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:



The following table describes this part's formal parameters:

1	Name		Туре		Mode		Description	1
Ī	Matrix	I	Matrices	I	0ut	I	Matrix to be made into an identity matrix	1

3.3.6.2.9.5.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name	1	Туре		Value		Description	Ī
	Col		Col_Indices		N/A		Index into second dimension of input matrix	
İ	Row	İ	Row_Indices	İ	N/A		Index into first dimension of inut matrix	

3.3.6.2.9.5.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.2.6 PROCESSING

```
The following describes the processing performed by this part:

procedure Set_To_Identity_Matrix (Matrix : out Matrices) is
```

```
-- --declaration section-
```

Col : Col_Indices;
Row : Row_Indices;

-- --begin procedure Set_to_Identity-

begin

-- -- make sure input matrix is a square matrix if Matrix'LENGTH(1) = Matrix'LENGTH(2) then

Matrix := (others => (others => 0.0));
Row := Matrix'FIRST(1);

Col := Matrix'FIRST(2);

3.3.6.2.9.5.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Unconstrained:

Name	Type	Description	Ī
Elements	floating point type discrete type	Data type of elements in exported matrix type Used to dimension exported matrix type	
Row_Indices	discrete type	Used to dimension exported matrix type	i

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_- Unconstrained:

Name	Ī	Range		Description
Matrices		N/A		Unconstrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Ī	Name	Description	
	Dimension_Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.5.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name		When/W	hy	Raise	i				Ī
[Dimension_Error		Raised	if	input	matrix	is	a square	matrix	

3.3.6.2.9.5.10.3 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P393-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.5.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī			Туре	1	Mode	1	Description	
Ī	Matrix						Matrix to be zeroed out	



3.3.6.2.9.5.10.3.4 LOCAL DATA

None.

3.3.6.2.9.5.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
begin

Matrix := (others => (others => 0.0));
end Set_To_Zero_Matrix;

3.3.6.2.9.5.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Unconstrained:

Name	Type	Description	1
Elements	floating point type	Data type of elements in exported matrix type	
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type	İ

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_- Unconstrained:

]	Name	Range	Description	1
	Matrices	N/A	Unconstrained, two-dimensional array of Elements	



3.3.6,2.9.5.10.3.8 LIMITATIONS

None.

3.3.6.2.9.5.10.4 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P394-0)

This function adds an input matrix to an identity matrix, returning the resultant matrix. The addition is performed by adding 1.0 to each diagonal element of the input matrix.

3.3.6.2.9.5.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	1	Туре	1	Mode	1	Description					ı
Input	1	Matrices	1	In	1	atrix to be adde	l to a	an	identity	matrix	Ī

3.3.6.2.9.5.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	1	Туре	Value	1	Description	l
Answer		Matrices	N/A		Result of adding identity matrix to input matrix	Ī
Col		Col Indices		ĺ	Index into second dimension of matrices	Ĺ
Row	<u> </u>	Row_Indices	N/A	 	Index into first dimension of matrices	

3.3.6.2.9.5.10.4.5 PROCESS CONTROL

Not applicable.

ore more ancestral units:

```
-36.7
```

```
3.3.6.2.9.5.10.4.6 PROCESSING
The following describes the processing performed by this part:
   function Add to Identity (Input: Matrices) return Matrices is
      --declaration section
      Answer : Matrices(Input'RANGE(1), Input'RANGE(2));
Col : Col_Indices;
Row : Row_Indices;
-- -- begin function Add_to_Identity
   begin
      --make sure input matrix is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then
         Answer := Input;
         Row := Input'FIRST(1);
         Col := Input'FIRST(2);
         Access Diagonal Elements:
             loop
                Answer(Row, Col) := Answer(Row, Col) + 1.0;
                exit Access Diagonal Elements when Row = Input'LAST(1);
                Row := Row Indices'SUCC(Row);
                Col := Col Indices'SUCC(Col);
            end loop Access Diagonal Elements;
      else
         -- do not have a square matrix
         raise Dimension Error;
      end if;
      return Answer;
   end Add to Identity;
3.3.6.2.9.5.10.4.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
The following tables describe the elements used by this part but defined in one
```

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_Unconstrained:

	Name	Range	Description	-
	Matrices	N/A	Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

Name	1	Description	<u>-</u>
Dimension_Error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.5.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	ı	When/Why	Raised	<u> </u>
Dimension_Error	I	Raised if	input matrix is not a square matrix	Ī

3.3.6.2.9.5.10.5 SUBTRACT FROM IDENTITY UNIT DESIGN (CATALOG #P395-0)

This function subtracts an input matrix from an identity matrix, returning the resultant matrix.

3.3.6.2.9.5.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.5.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description	Ī
Input	Matrices	In	Matrix to be subtracted from an identity matrix	

3.3.6.2.9.5.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Туре	Value	Description
Answer	Matrices	N/A	Result of subtracting input matrix from an identity matrix
Col	Col_Indices	N/A	Used to index second dimension of matrices
Col_Count	POSITIVE 	N/A 	Used to count the number of columns accessed; needed to determine when the diagonal element has been reached
Row	Row_Indices	i n/a i	Used to index second dimension of matrices
Row_Count	POSITIVE	N/A	Used to count the number of rows accessed; when Col_Count equals this the diagonal element has been reached
S_Col 	Col_Indices	N/A	Used to mark the column containing the diagonal element for the current row
S_Row	Row_Indices	N/A 	Used in conjunction with S_Col to locate the symmetric counterpart to the element being referenced in the bottom half of the matrix

///

```
3.3.6.2.9.5.10.5.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.5.10.5.6 PROCESSING
The following describes the processing performed by this part:
   function Subtract from Identity (Input : Matrices) return Matrices is
      --declaration section
     Answer : Matrices(Input'RANGE(1), Input'RANGE(2));
Col : Col_Indices;
     Col_Count : POSTTIVE;
     Row : Row Indices;
     Row_Count : POSITIVE;
      S_Col : Col_Indices;
     S Row : Row Indices;
-- --begin function Subtract from Identity
  begin
     -- make sure input matrix is a square matrix
     if Input'LENGTH(1) = Input'LENGTH(2) then
        --will subtract input matrix from an identity matrix by first
        --subtracting all elements from 0.0 and then adding 1.0 to the
        --diagonal elements;
        --when doing the subtraction, will only calculate the remainder
        --for the elements in the bottom half of the matrix and will simply
        --do assignments for the symmetric elements in the top half of the
        --matrix
        Row Count := 1;
        --S Col will go across the columns as Row goes down the rows;
        --will mark column containing the diagonal element for this row
        Row := Input'FIRST(1);
        S Col := Input'FIRST(2);
        Do Every Row:
          loop
              Col Count := 1;
              --S Row will go down the rows as Col goes across the columns;
              --when paired with S Col will mark the symmetric counterpart
              -- to the element being referenced in the bottom half of the
              --matrix
              Col
                        := Input'FIRST(2);
              S Row
                        := Input'FIRST(1);
```

```
Subtract_Elements_From_Zero:
                  loop
                     --perform subtraction on element in bottom half of matrix
                     Answer(Row,Col) := - Input(Row,Col);
                     --exit loop after diagonal element has been reached
                     exit Subtract Elements From Zero when Col Count =
                                                               Row Count;
                     --assign values to symmetric elements in top half of matrix
                     -- (done after check for diagonal, since diagonal elements
                     -- don't have a symmetric counterpart)
                     Answer(S_Row,S_Col) := Answer(Row,Col);
                     --increment variables
                     Col Count := Col Count + 1;
                              := Col Indices'SUCC(Col);
                     Col
                               := Row Indices'SUCC(S Row);
                     S Row
                  end loop Subtract Elements From Zero;
               --add one to the diagonal element
               Answer(Row, Col) := Answer(Row, S Col) + 1.0;
               exit Do Every_Row when Row_Count = Input'LENGTH(1);
              Row Count := Row Count + 1;
                         := Row Indices'SUCC(Row);
               Row
                         := Col Indices'SUCC(S Col);
               S_{Col}
           end loop Do Every Row;
     else
        raise Dimension Error;
     end if;
     return Answer;
  end Subtract from Identity;
3.3.6.2.9.5.10.5.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
```

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric Full Storage Matrix -Operations Unconstrained:





Name	Type	Description	
Elements	floating point type	Data type of elements in exported matrix type	
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type	

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:

Ī	Name		Range	1	Description	
	Matrices		N/A		Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

	Name	1	Description	1
	Dimension_Error	i	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.5.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	1	When/Wh	Rais	ed			
Dimension_Error					 	 	

3.3.6.2.9.5.10.6 "+" UNIT DESIGN (CATALOG #P396-0)

This function adds two symmetric matrices by adding the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.5.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.



3.3.6.2.9.5.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	1	Туре	1	Mode		Description
	Left Right		Matrices Matrices		In In		First matrix to be added Second matrix to be added

3.3.6.2.9.5.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding input matrices
Row_Count, Col_Count	Positive 	N/A 	Used to keep track of the number of rows and columns which have been handled
L Col	Col Indices	N/A	Column index into left matrix
L Row	Row Indices	N/A	Row index into left matrix
R Col	Col Indices	N/A	Column index into right matrix
R Row	Row Indices	N/A	Row index into right matrix
S_Col	Col_Indices	N/A	New column index after row and column have been swapped, i.e. (i,j) => (j,i)
S_Row	Row_Indices	N/A	New row index after row and column have been swapped

3.3.6.2.9.5.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.6.6 PROCESSING

The following describes the processing performed by this part:

function "+" (Left : Matrices;

Right: Matrices) return Matrices is





```
--declaration section
      Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
      Col Count : POSITIVE;
      Row_Count : POSITIVE;
      L_Col : Col_Indices;
L_Row : Row_Indices;
R_Col : Col_Indices;
R_Row : Row_Indices;
S_Col : Col_Indices;
S_Row : Row_Indices;
S_Row : Row_Indices;
-- --begin function "+"
__ _____
   begin
      --make sure both input matrices are square matrices of the same size
      if Left'LENGTH(1) = Left'LENGTH(2) and
         Left'LENGTH(1) = Right'LENGTH(1) and
         Right'LENGTH(1) = Right'LENGTH(2) then
         --addition calculations will only be carried out on the bottom half
         --of the input matrices followed by assignments to the symmetric
         --elements in the top half of the matrix
         Row Count := 1;
         --as L Row goes down the rows, S Col will go across the columns
         L_Row := Left'FIRST(1);
         S Col
                   := Left'FIRST(2);
         R Row
                   := Right'FIRST(1);
         Do All Rows:
             loop
                Col Count := 1;
                --as L_Col goes across the columns, S_Row will go down the rows
                L Col := Left'FIRST(2);
                S Row := Left'FIRST(1);
                R Col := Right'FIRST(2);
                Add Bottom Half Elements:
                   Toop
                      Answer(L Row, L Col) := Left(L Row, L Col) +
                                                Right(\overline{R} Row, \overline{R} Col);
                      --exit when diagonal element has been reached
                      exit Add Bottom Half Elements when Col Count = Row Count;
                      --assign value to symmetric element in top half of matrix
                      --(do this after exit since diagonal elements don't have
                      -- a corresponding symmetric element)
```

```
Answer(S_Row, S_Col) := Answer(L_Row, L_Col);
                --increment values
                Col Count := Col Count + 1;
                         := Col Indices'SUCC(L Col);
                S Row
                         := Row Indices'SUCC(S Row);
                R_Col
                          := Col Indices'SUCC(R Col);
             end loop Add Bottom Half Elements;
         exit Do All Rows when Row Count = Left'LENGTH(1);
         Row Count := Row Count + \overline{1};
                   := Row_Indices'SUCC(L_Row);
:= Col_Indices'SUCC(S_Col);
         L Row
         S_Col
         R Row
                    := Row Indices'SUCC(R Row);
      end loop Do All Rows;
else
   raise Dimension Error;
end if;
```

3.3.6.2.9.5.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

return Answer;

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

end "+";

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Unconstrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Unconstrained:





Ī	Name	1	Range	1	Description	
	Matrices		N/A		Unconstrained, two-dimensional array of Elements	

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

1	Name		Description	
	Dimension_	Error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.5.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

1	Name	1	When/W	hy	Raised	l							1
1	Dimension_Error		Raised	if	input	matrices	are	not	both	m x	m	matrices	1

3.3.6.2.9.5.10.7 "-" UNIT DESIGN (CATALOG #P397-0)

This function subtracts two symmetric input matrices by subtracting the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.5.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

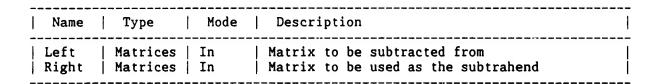
3.3.6.2.9.5.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.5.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:



3.3.6.2.9.5.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding input matrices
Row_Count, Col_Count	Positive	N/A	Used to keep track of the number of rows and columns which have been handled
L Col	Col Indices	N/A	Column index into left matrix
L_Row	Row Indices	N/A	Row index into left matrix
R Col	Col Indices	N/A	Column index into right matrix
R Row	Row Indices	N/A	Row index into right matrix
S_Col	Col_Indices	N/A	New column index after row and column have been swapped, i.e. (i,j) => (j,i)
S_Row	Row_Indices	N/A	New row index after row and column have been swapped

3.3.6.2.9.5.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.5.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
function "-" (Left : Matrices;
```

Right: Matrices) return Matrices is

```
-- --declaration section
```

```
Answer : Matrices(Left'RANGE(1), Left'RANGE(2));
Col_Count : POSITIVE;
Row_Count : POSITIVE;
L_Col : Col_Indices;
L_Row : Row_Indices;
R_Col : Col_Indices;
R_Row : Row_Indices;
S_Col : Col_Indices;
```

```
S Row : Row Indices;
-- --begin function "+"
__ ___
   begin
      --make sure both input matrices are square matrices of the same size
      if Left'LENGTH(1) = Left'LENGTH(2) and
   Left'LENGTH(1) = Right'LENGTH(1) and
         Right'LENGTH(1) = Right'LENGTH(2) then
         --addition calculations will only be carried out on the bottom half
         -- of the input matrices followed by assignments to the symmetric
         --elements in the top half of the matrix
        Row Count := 1;
         --as L Row goes down the rows, S Col will go across the columns
         L_Row := Left'FIRST(1);
                  := Left'FIRST(2);
         S Col
         R Row
                 := Right'FIRST(1);
         Do All Rows:
            loop
               Col Count := 1;
               --as L Col goes across the columns, S Row will go down the rows
               L Col := Left'FIRST(2);
               S Row := Left'FIRST(1);
               R Col := Right'FIRST(2);
               Add Bottom Half Elements:
                  Тоор
                     Answer(L Row, L Col) := Left(L Row, L Col) -
                                            Right(R Row, R Col);
                     --exit when diagonal element has been reached
                     exit Add_Bottom_Half_Elements when Col_Count = Row_Count;
                     --assign value to symmetric element in top half of matrix
                     -- (do this after exit since diagonal elements don't have
                     -- a corresponding symmetric element)
                     Answer(S Row, S Col) := Answer(L Row, L Col);
                     --increment values
                     Col Count := Col Count + 1;
                     L_Col := Col_Indices'SUCC(L_Col);
                     S Row
                             := Row_Indices'SUCC(S_Row);
                     R Col := Col Indices'SUCC(R Col);
                  end loop Add Bottom Half Elements;
               exit Do All Rows when Row Count = Left'LENGTH(1);
```



```
Row Count := Row Count + 1;
                   := Row_Indices'SUCC(L_Row);
            L Row
             S<sup>C</sup>ol
                      := Col Indices'SUCC(S Col);
            R Row
                      := Row Indices' SUCC(R Row);
         end loop Do All Rows;
   else
      raise Dimension Error;
   end if:
   return Answer;
end "-";
```

3.3.6.2.9.5.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:



The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Unconstrained:

Ī	Name	Type	 	Description	Ī
	Elements	floating point type		Data type of elements in exported matrix type	
	Col_Indices Row_Indices	discrete type discrete type		Used to dimension exported matrix type Used to dimension exported matrix type	

The following table summarizes the types required by this part and defined in the package specification of Symmetric Full Storage Matrix Operations -Unconstrained:

1	Name		Range		Description	1
	Matrices		N/A		Unconstrained, two-dimensional array of Elements	

Exceptions:



The following table describes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description	
_	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.5.10.7.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	1	When/V	/hy	Raise	 i								
Dimension_Error		Raised	if	input	matrices	are	not	both	m	 х 	m m	natrices	

3.3.6.2.9.6 DIAGONAL MATRIX OPERATIONS PACKAGE DESIGN (CATALOG #P408-0)

This package contains a set of functions designed to operate on a diagonal matrix.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.2 LOCAL ENTITIES DESIGN

Subprograms:

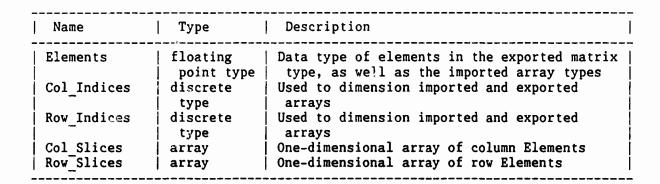
This package body contains a sequence of statements which are executed when it is elaborated. This code first checks to ensure a square matrix has been instantiated. If not, a Dimension Error exception is raised. If a square matrix has been instantiated, then the Row Marker and Col Marker arrays are initialized.

3.3.6.2.9.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters are defined in this part's package specification:

Data types:



3.3.6.2.9.6.4 LOCAL DATA

Exceptions:

The following table describes the exceptions defined by this part's package specification:

]	Name	Ī	Description	-
	Invalid_Index		Indicates an attempt was made to access an element not on the diagonal	

Data types:

The following table describes the data types defined in this part's package specification:

Name	Range	Description	
Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices	
	N/A	Vector representation of a matrix where all but the diagonal elements equal zero	

The following table describes the data types defined by this part:

Name	Range	Description
Row_Markers	N/A	Array of pointers into Diagonal_ Matrices
Col_Markers	N/A	Array of pointers into Diagonal_ Matrices

Data objects:



The following table describes the data objects defined in this part's package specification:

Name		Туре		Descripti	on					
Entry_Count		Positive	1	Number of	diagonal	elements	in	the	array	I

The following table describes the data objects maintained by this part:

Name	Type	Description
Row_Marker	Row_Markers	Array of pointers into Diagonal_ Matrices
CoJ_Marker	Col_Markers	Array of pointers into Diagonal_ Matrices
Row_Minus_Col_Indices_ Pos_First	INTEGER	Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)
Local_Identity_Matrix	Diagonal_ Matrices	Pre-Initialized identity matrix
Local_Zero_Matrix	Diagonal_ Matrices	Pre-initialized zero matrix

The following table describes the data objects contained in the declare block located at the end of this package body:

Ī	Name	l	Туре		Value		Description	1
	Col_Count Row_Count		Positive Positive		N/A N/A		Counts the number of columns Counts the nubmer of rows	

3.3.6.2.9.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Diagonal_Matrix_Operations is

```
-- --local declarations
```

type Col_Markers is array(Col_Indices) of POSITIVE; type Row_Markers is array(Row_Indices) of POSITIVE;

```
Col Marker : Col Markers;
   Row Marker: Row Markers;
   Row Minus Col Indices Pos First : constant INTEGER
                                     := Row_Indices'POS(Row_Indices'FIRST) -
                                       Col Indices'POS(Col Indices'FIRST);
   Local Identity Matrix : constant Diagonal Matrices := (others => 1.0);
   Local Zero Matrix : constant Diagonal Matrices := (others => 0.0);
______
--begin processing for Diagonal
--Matrix Operations package
begin
   Init Block:
     declare
        Col Count : POSITIVE;
        Row Count : POSITIVE;
     begin
        --make sure lengths of indices are the same
        if Row Slices'LENGTH = Col Slices'LENGTH then
           --initialize row and column marker arrays
           Row Count := 1;
           Row Init:
              for Row in Row Indices loop
                 Row Marker(Row) := Row Count;
                                 := Row Count + 1;
                 Row Count
              end loop Row Init;
           Col Count := 1;
           Col Init:
              For Col in Col Indices loop
                 Col Marker(Col) := Col Count;
                 Col Count
                                := Col Count + 1;
              end loop Col Init;
        else
           raise Dimension Error;
        end if;
     end Init_Block;
end Diagonal Matrix Operations;
```

3.3.6.2.9.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

[Name		Description	
	Dimension_Error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name		When/Why	Raised				
Dimension	Error	Raised if not the		of Row_Indi	ces and Co	l_Indices ar	e

3.3.6.2.9.6.9 LLCSC DESIGN

None.

3.3.6.2.9.6.10 UNIT DESIGN

3.3.6.2.9.6.10.1 IDENTITY MATRIX UNIT DESIGN (CATALOG #P409-0)

This function returns a diagonal matrix which has been set to an identity matrix. An identity matrix is one where all the elements equal 0.0, except for the diagonal elements which equal 1.0.

3.3.6.2.9.6.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.1.2 LOCAL ENTITIES DESIGN

None.

19.

3.3.6.2.9.6.10.1.3 INPUT/OUTPUT

None.

3.3.6.2.9.6.10.1.4 LOCAL DATA

None.

3.3.6.2.9.6.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.1.6 PROCESSING

The following describes the processing performed by this part:

function Identity_Matrix return Diagonal_Matrices is

begin

return Local Identity Matrix;

end Identity Matrix;

3.3.6.2.9.6.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:



Ī	Name	Range	Description	<u> </u>
Ī	Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices	
j		N/A	Vector representation of a matrix where all but the diagonal elements equal zero	İ

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal Matrix Operations:

Ī	Name	Type	Description
	Local_Identity_Matrix	Diagonal_ Matrices	Pre-initialized identity matrix

3.3.6.2.9.6.10.1.8 LIMITATIONS

None.

3.3.6.2.9.6.10.2 ZERO_MATRIX UNIT DESIGN (CATALOG #P410-0)

This function returns a diagonal matrix which has been set to a zero matrix.

3.3.6.2.9.6.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.2.3 INPUT/OUTPUT

None.

3.3.6.2.9.6.10.2.4 LOCAL DATA

None.

3.3.6.2.9.6.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.2.6 PROCESSING

The following describes the processing performed by this part:

function Zero_Matrix return Diagonal_Matrices is

begin

return Local Zero Matrix;

end Zero Matrix;

3.3.6.2.9.6.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Name	Туре	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types:
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_Range	1Entry_Count	Used to dimension diagonal_matrices
Diagonal Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

. The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:



Name	Type	Description
Local_Zero_Matrix	Diagonal_ Matrices	Pre-initialized zero matrix

3.3.6.2.9.6.10.2.8 LIMITATIONS

None.

3.3.6.2.9.6.10.3 CHANGE_ELEMENT UNIT DESIGN (CATALOG #P411-0)

This procedure changes a single element of a diagonal matrix.

3.3.6.2.9.6.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Value Row Col Matrix	Elements Row_Indices Col_Indices Diagonal Matrices	In In In Out	New value to be placed in matrix Row where value is to be placed Column where value is to be placed Diagonal matrix to be updated

3.3.6.2.9.6.10.3.4 LOCAL DATA

None.

3.3.6.2.9.6.10.3.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.6.10.3.6 PROCESSING

The following describes the processing performed by this part:

Matrix : out Diagonal Matrices) is

begin

-- --make sure element referenced is on the diagonal if Row_Marker(Row) = Col_Marker(Col) then

Matrix(Row_Marker(Row)) := New_Value;

else

raise Invalid_Index;

end if;

3.3.6.2.9.6.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

end Change Element;

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal Matrix Operations:



Name	Range	Description
Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal Matrix_Operations:

Name Range	Description
Row_Markers N/A	Array of pointers into Diagonal_ Matrices
Col_Markers N/A	Array of pointers into Diagonal_ Matrices

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Type	Description
Row_Marker	Row_Markers	Array of pointers into Diagonal_ Matrices
Col_Marker	Col_Markers	Array of pointers into Diagonal_ Matrices

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Description	
Invalid_Index	Indicates an attempt was made to access an element not or the diagonal	1

3.3.6.2.9.6.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

1	Name	I	When/Why	Raised									I
	Invalid_Index		Raised if diagonal	in Row	and	Col	indices	do	not	fall	on	the	

3.3.6.2.9.6.10.4 RETRIEVE ELEMENT UNIT DESIGN (CATALOG #P412-0)

This function returns an element contained in a diagonal matrix.

3.3.6.2.9.6.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	Ī	Туре		Mode	1	Description
Ma	atrix		Diagonal_ Matrices		In		Diagonal matrix containing element desired
Re	ow ol	İ	Row_Indices Col_Indices		In In	İ	Row containing element desired Column containing element desired

3.3.6.2.9.6.10.4.4 LOCAL DATA

None.

3.3.6.2.9.6.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.4.6 PROCESSING

The following describes the processing performed by this part:

function Retrieve Element (Matrix : Diagonal Matrices;

: Row Indices; Row

Col : Col_Indices) return Elements is

begin

--make sure (row, col) falls on the diagonal if Row Marker(Row) /= Col Marker(Col) then raise Invalid_Index; end if;

return Matrix(Row Marker(Row));

end Retrieve Element;

3.3.6.2.9.6.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal Matrix Operations:

Name	Range	Description
Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices
Diagonal Matrices	N/A 	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal Matrix Operations:

Name Range	Description	1
Row_Markers N/A	Array of pointers into Diagonal_ Matrices	-
Col_Markers N/A	Array of pointers into Diagonal_ Matrices	

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal Matrix Operations:



Name	Type	Description
Row_Marker	Row_Markers	Array of pointers into Diagonal_
Col_Marker	Col_Markers	Array of pointers into Diagonal_ Matrices

3.3.6.2.9.6.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	١	When/Why	Raised			1
Invalid_Index						 diagonal

3.3.6.2.9.6.10.5 ROW SLICE UNIT DESIGN (CATALOG #P413-0)

This function returns a row of values from a diagonal matrix.

3.3.6.2.9.6.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Nawe		Туре		Mode		Description	1
-	Matrix		Diagonal_ Matrices Row_Indices		In		Diagonal matrix containing row desired	
ļ	Row	i	Row_Indices		In		Row desired	i

3.3.6.2.9.6.10.5.4 LOCAL DATA

Data objects:



The following table describes the data objects maintained by this part:

-	Name	1	Туре	Value	1	Description	
- 	Col_Spot Answer		Col_Indices Row_Slices	N/A N/A		Index into Answer Row of values	

3.3.6.2.9.6.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
function Row Slice (Matrix : Diagonal Matrices;
                            : Row Indices) return Row Slices is
                       Row
     --declaration section
     Col Spot : Col Indices;
      Answer : Row Slices;
-- --begin function Row_Slice
  begin
     --zero out slice
     Answer := (others \Rightarrow 0.0);
     --insert diagonal element
     Col Spot := Col Indices'VAL(Row Indices'POS(Row) -
                                  Row Minus Col Indices Pos First);
     Answer(Col Spot) := Matrix(Row Marker(Row));
     return Answer;
  end Row Slice;
```

3.3.6.2.9.6.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:



Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays
Row_Slices	array	One-dimensional array of row Elements

The following table summarizes the types required by this part and defined in the package specification of Diagonal Matrix Operations:

Name	Range	Description
Diagonal_Range	1Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal Matrix Operations:

Ī	Name	I	Range	Ī	Operators		Description	Ī
	Row_Markers		N/A		N/A		Array of pointers into Diagonal_ Matrices	

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal_Matrix_Operations:

Name	Туре	Description
Col_Marker Row_Minus_Col_Indices_	Col_Markers INTEGER	Array of pointers into Diagonal_
Pos_First		Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)



3.3.6.2.9.6.10.5.8 LIMITATIONS

None.

3.3.6.2.9.6.10.6 COLUMN SLICE UNIT DESIGN (CATALOG #P414-0)

This function returns a column's worth of values from a diagonal matrix.

3.3.6.2.9.6.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	Ī	Туре	l	Mode	Ī	Description]
	Matrix Col		Diagonal_ Matrices Col_Indices		In In In		Diagonal matrix containing desired column of values Indicates which column is desired	

3.3.6.2.9.6.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name	1	Туре	I	Value		Description	
	Answer Row_Spot		Col_Slices Row_Indices		N/A N/A		Column of values from input matrix Index into Answer	

3.3.6.2.9.6.10.6.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.6.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
function Column_Slice (Matrix : Diagonal_Matrices;
Col : Col_Indices) return Col_Slices is
```

```
-- --declaration section
```

```
Answer : Col_Slices;
Row_Spot : Row_Indices;
```

```
-- --begin function Column_Slice
```

begin

```
-- --zero out answer and then insert diagonal value
Answer := (others => 0.0);
```

```
-- --insert diagonal value

Row_Spot := Row_Indices'VAL(Col_Indices'POS(Col) +

Row_Minus_Col_Indices_Pos_First);

Answer(Row_Spot) := Matrix(Col_Marker(Col));
```

return Answer;

end Column Slice;

3.3.6.2.9.6.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Ī	Name	Type	Description
	Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
	Col_Indices	discrete type	Used to dimension imported and exported arrays
İ	Row_Indices	discrete type	Used to dimension imported and exported arrays
İ	Col_Slices	array	One-dimensional array of column Elements



The following table summarizes the types required by this part and defined in the package specification of Diagonal Matrix Operations:

Name	Range	Description
Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices
_	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

The following table summarizes the types required by this part and defined in the package body of Diagonal Matrix Operations:

Name Range	e Description
Col_Markers N/A	Array of pointers into Diagonal_ Matrices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Diagonal Matrix Operations:

]	Name	Туре	Description
Ī	Col_Marker	Col_Markers	Array of pointers into Diagonal_ Matrices
	Row_Minus_Col_Indices_ Pos_First	INTEGER	Preinitialized value of: Row_Indices'POS(Row_Indices'FIRST) - Col_Indices'POS(Col_Indices'FIRST)

3.3.6.2.9.6.10.6.8 LIMITATIONS

None.

3.3.6.2.9.6.10.7 ADD TO IDENTITY UNIT DESIGN (CATALOG #P415-0)

This function adds the input matrix to an identity matrix, returning the resultant diagonal matrix. The calculations are performed by adding 1.0 to each diagonal element.

3.3.6.2.9.6.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name]	Туре		Mode	1	Description	
	Input		Diagonal_Matrices		In		Diagonal matrix to be added to an identity matrix.	

3.3.6.2.9.6.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name Type	Value	Description	Ī
Answer Diagonal_Matrice	s N/A	Result of performing addition	Ī

3.3.6.2.9.6.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.7.6 PROCESSING

The following describes the processing performed by this part:

function Add_to_Identity (Input : Diagonal_Matrices) return Diagonal_Matrices is

-- --declaration section

Answer : Diagonal Matrices;

-- --begin function Add_to_Identity

begin

Process:

```
for Index in 1..Entry Count loop
   Answer(Index) := Input(Index) + 1.0;
end loop Process;
```

return Answer;

end Add to Identity;

3.3.6.2.9.6.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices
Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal Matrix Operations:

Ī	Name		Туре	•	Description	•	 	
Ī	Entry_Count	1	Positive	1	Number of diagona	elements	 	 1

3.3.6.2.9.6.10.7.8 LIMITATIONS

None.

3.3.6.2.9.6.10.8 SUBTRACT FROM IDENTITY UNIT DESIGN (CATALOG #P416-0)

This function subtracts an input matrix from an identity matrix, returning the result. The calculations are performed by subtracting the diagonal elements from 1.0.

3.3.6.2.9.6.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.8.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Туре		Mode	Ī	Description	-
	Input		Diagonal_Matrices		In	1	Diagonal matrix to be subtracted from an identity matrix	

3.3.6.2.9.6.10.8.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name Type	Value	Description
Answer Diagonal_Matrices	N/A	Result of performing the subtraction

3.3.6.2.9.6.10.8.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.6.10.8.6 PROCESSING

```
The following describes the processing performed by this part:
```

```
-- --declaration section
```

```
Answer : Diagonal Matrices;
```

```
-- --begin function Subtract_From_Identity
```

begin

```
Process:
```

```
for Index in 1..Entry Count loop
   Answer(Index) := 1.0 - Input(Index);
end loop Process;
```

return Answer;

end Subtract from Identity;

3.3.6.2.9.6.10.8.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal_Matrix_Operations:

Name	Туре	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:



- 	Name	Range	Description	
	Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices	
	Diagonal_ Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero	İ

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal_Matrix_Operations:

	· — ·	Name		Туре		Descrip	tion					
ا]	Entry_Count		Positive		Number of	f diagonal	elements	in	the	array	

3.3.6.2.9.6.10.8.8 LIMITATIONS

None.

3.3.6.2.9.6.10.9 "+" (DIAGONAL MATRICES + DIAGONAL MATRICES => DIAGONAL MATRICES)
UNIT DESIGN (CATALOG #P417-0)

This function adds two input diagonal matrices, returning the resultant diagonal matrix.

3.3.6.2.9.6.10.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.6.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.9.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Туре		Mode		Description	Ī
	Left Right		Diagonal_Matrices Diagonal_Matrices		In In		First diagonal matrix to be added Second diagonal matrix to be added	



3.3.6.2.9.6.10.9.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name Type	Value	Description	Ī
Answer Diagonal_Matrices	N/A	Result of performing the addition	

3.3.6.2.9.6.10.9.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.9.6 PROCESSING

The following describes the processing performed by this part:

-- --declaration section

Answer : Diagonal_Matrices;

```
--- --begin function "+"
```

begin

```
Process:
    for Index in 1..Entry_Count loop
        Answer(Index) := Left(Index) + Right(Index);
    end loop Process;
return Answer;
```

end "+";

3.3.6.2.9.6.10.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:



The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Name	Type	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal Matrix Operations:

Name	Range	Description	<u>-</u>
Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices	
Diagonal Matrices	N/A	Vector representation of a matrix where all but the diagonal elements equal zero	

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Type	Description	<u> </u>
Entry_Count	Positive	Number of diagonal elements in the array	1

3.3.6.2.9.6.10.9.8 LIMITATIONS

None.

3.3.6.2.9.6.10.10 "-" (DIAGONAL MATRICES - DIAGONAL MATRICES -> DIAGONAL MATRICES)
UNIT DESIGN (CATALOG #P418-0)

This function subtracts two input diagonal matrices, returning the resultant matrix. The calculations are performed by subtracting the individual elements of the input matrices.

3.3.6.2.9.6.10.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.



3.3.6.2.9.6.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.6.10.10.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Ī	Туре	Ī	Mode	 	Description				
Left Right 		Diagonal_Matrices Diagonal_Matrices		In In		Diagonal matrix Diagonal matrix subtrahend	to to	be be	subtracted treated as	from the

3.3.6.2.9.6.10.10.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

•	Name	•		1	Value		Description
			Diagonal_Matrices			1	Result of performing the subtraction

3.3.6.2.9.6.10.10.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.6.10.10.6 PROCESSING

The following describes the processing performed by this part:

-- --begin function "-"

begin



```
Process:
    for Index in 1..Entry_Count loop
        Answer(Index) := Left(Index) - Right(Index);
    end loop Process;

return Answer;
end "-";
```

3.3.6.2.9.6.10.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following generic data types are visible to this part and defined at the package specification level of Diagonal Matrix Operations:

Name	Туре	Description
Elements	floating point type	Data type of elements in the exported matrix type, as well as the imported array types
Col_Indices	discrete type	Used to dimension imported and exported arrays
Row_Indices	discrete type	Used to dimension imported and exported arrays

The following table summarizes the types required by this part and defined in the package specification of Diagonal_Matrix_Operations:

Name	Range	Description
Diagonal_ Range	1Entry_Count	Used to dimension diagonal_matrices
	N/A	Vector representation of a matrix where all but the diagonal elements equal zero

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Diagonal Matrix Operations:

Name		Туре	Ī	Description	Ī
Entry_Count	Ī	Positive	 	Number of diagonal elements in the array	



3.3.6.2.9.6.10.10.8 LIMITATIONS

None.

3.3.6.2.9.7 VECTOR_SCALAR_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P419-0)

This package provides a set of functions to multiply and divide each element of a vector by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.7.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the units in this part:

 Name	Requirements Allocation	
#*# #/#	R065 R066	

3.3.6.2.9.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.7.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types previously in this part's package specification:



1	Name	Type	Description
-	Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
İ	Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
j	Scalars	floating point type	Type of value to be used for multiplying and dividing
	Indices1	discrete type	Used to dimension Vectors1
İ	Indices2	discrete type	Used to dimension Vectors2
İ	Vectors1	array	An array of Elements1
İ	Vectors2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars
"/" 	function	Used to define the operation Elements2 := Elements1 / Scalars

3.3.6.2.9.7.4 LOCAL DATA

None.

3.3.6.2.9.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.7.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Vector_Scalar_Operations_Unconstrained is

end Vector_Scalar_Operations_Unconstrained;

3.3.6.2.9.7.7 UTILIZATION OF OTHER ELEMENTS





3.3.6.2.9.7.8 LIMITATIONS

None.

3.3.6.2.9.7.9 LLCSC DESIGN

None.

3.3.6.2.9.7.10 UNIT DESIGN

3.3.6.2.9.7.10.1 "*" UNIT DESIGN (CATALOG #P420-0)

This function calculates a scaled vector by multiplying each element of an input vector by a scale factor.

3.3.6.2.9.7.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO65.

3.3.6.2.9.7.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.7.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	1	Туре	I	Mode		Description
Vector Multiplier		Vectors2 Scalars	•	In In		Vector to be scaled Scale factor

3.3.6.2.9.7.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name	1	Туре	1	Description	
	A Index		Indices1		Scaled vector Index into answer array Index into input vector	

```
3.3.6.2.9.7.10.1.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.7.10.1.6 PROCESSING
The following describes the processing performed by this part:
   function "*" (Vector
                        : Vectors2;
                 Multiplier : Scalars) return Vectors1 is
      --declaration section-
     Answer : Vectors1(Indices1'FIRST ...
                         Indices1'VAL(Vector'LENGTH-1 +
                                      Indices1'POS(Indices1'FIRST) ));
     A Index : Indices1;
     V Index : Indices2;
-- --begin function "*"
  begin
    A Index := Indices1'FIRST;
    V Index := Indices2'FIRST;
    Process:
       loop
          Answer(A Index) := Vector(V Index) * Multiplier;
          exit Process when V Index = Vector'LAST;
          A Index := Indices1\(^T\)SUCC(A Index);
          V_Index := Indices2'SUCC(V_Index);
       end loop Process;
    return Answer;
  end "*";
```

3.3.6.2.9.7.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_-Unconstrained:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices1	discrete type	Used to dimension Vectors1
Indices2	discrete type	Used to dimension Vectors2
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Vector_Scalar_Operations_Unconstrained package:

Name	Type	Description	
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars	

3.3.6.2.9.7.10.1.8 LIMITATIONS

None.

3.3.6.2.9.7.10.2 "/" UNIT DESIGN (CATALOG #P421-0)

This function calculates a scaled vector by dividing each element of an input vector by a scale factor.

3.3.6.2.9.7.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO66.

3.3.6.2.9.7.10.2.2 LOCAL ENTITIES DESIGN

None.



3.3.6.2.9.7.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Туре		Mode	Ī	Description
	Vector Divisor		Vectors1 Scalars		In In		Vector to be scaled Scale factor

3.3.6.2.9.7.10.2.4 LOCAL DATA

Data objects:

The following describes the local data maintained by this part:

	Name		Туре	1	Description
İ	A_Index		Indices2	1	Scaled vector Index into answer vector Index into input vector

3.3.6.2.9.7.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.7.10.2.6 PROCESSING

The following describes the processing performed by this part:

function "/" (Vector : Vectors1;

Divisor : Scalars) return Vectors2 is

--declaration section-

Answer : Vectors2(Indices2'FIRST ...

Indices2'VAL(Vector'LENGTH-1 +

Indices2'POS(Indices2'FIRST)));

A Index : Indices2; V Index : Indices1;

--begin function Vector Scalar Divide



```
A_Index := Indices2'FIRST;
V_Index := Indices1'FIRST;
Process:
    loop

    Answer(A_Index) := Vector(V_Index) / Divisor;
    exit Process when V_Index = Indices1'LAST;
    A_Index := Indices2'SUCC(A_Index);
    V_Index := Indices1'SUCC(V_Index);
    end loop Process;
return Answer;
end "/";
```

3.3.6.2.9.7.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this parc but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_- Unconstrained:

Name	Type	Description
Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
Scalars	floating point type	Type of value to be used for multiplying and dividing
Indices1	discrete type	Used to dimension Vectors1
Indices2	discrete type	Used to dimension Vectors2
Vectors1	array	An array of Elements1
Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Vector_Scalar_Operations_Unconstrained package:



Name	Type	Description	
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars	

3.3.6.2.9.7.10.2.8 LIMITATIONS

None.

3.3.6.2.9.8 MATRIX_SCALAR_OPERATIONS_UNCONSTRAINED PACKAGE DESIGN (CATALOG #P425-0)

This package provides a set of functions which will scale a matrix by multiplying or dividing each element of the matrix by a scale factor.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.8.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the parts in this LLCSC:

	Name		Requirements Allocation	
	n×n n/n		R073 R074	

3.3.6.2.9.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.8.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described in this part's package specification:

Data types:



Name	Type	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col Indices1	discrete type	Used to dimension second dimension of Matrices1
Row Indices1	discrete type	Used to dimension first dimension of Matrices1
Col Indices2	discrete type	Used to dimension second dimension of Matrices2
Row Indices2	discrete type	Used to dimension first dimension of Matrices2
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms:

Name	Туре	Description
n*u	function	Function to define the operation Elements1 * Scalars := Elements2
""/""	function	Function to define the operation Elements2 / Scalars := Elements1

3.3.6.2.9.8.4 LOCAL DATA

None.

3.3.6.2.9.8.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.8.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Matrix_Scalar_Operations_Unconstrained is

end Matrix_Scalar_Operations_Unconstrained;

 $\hat{\beta}\Phi$

3.3.6.2.9.8.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.8.8 LIMITATIONS

None.

3.3.6.2.9.8.9 LLCSC DESIGN

None.

3.3.6.2.9.8.10 UNIT DESIGN

3.3.6.2.9.8.10.1 "*" UNIT DESIGN (CATALOG #P426-0)

This function calculates a scaled matrix by multiplying each element of an input matrix by a scalar.

3.3.6.2.9.8.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R073.

3.3.6.2.9.8.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.8.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Type		Mode	1	Description
	Matrix Multiplier		Matrices1 Scalars		In In		Matrix to be scaled Scale factor

3.3.6.2.9.8.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:



Name	Type	Description	
Answer A_Col A_Row M_Col M_Row	Matrices2 Col_Indices2 Row_Indices2 Col_Indices1 Row_Indices1	Index into first dimension of answer matrix Index into second dimension of input matrix	

```
3.3.6.2.9.8.10.1.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.8.10.1.6 PROCESSING
The following describes the processing performed by this part:
   function "*" (Matrix
                         : Matrices1:
                 Multiplier : Scalars) return Matrices2 is
      --declaration section-
      Answer : Matrices2
                  (Row Indices2'FIRST ..
                   Row_Indices2'VAL(Matrix'LENGTH(1)-1 +
                                     Row Indices2'POS(Row Indices2'FIRST) ),
                   Col Indices2'FIRST ..
                   Col Indices2'VAL(Matrix'LENGTH(2)-1 +
                                     Col Indices2'POS(Col Indices2'FIRST) ));
      A Col : Col Indices2;
      A_Row : Row_Indices2;
M_Col : Col_Indices1;
      M_Row : Row Indices1;
-- --begin function "*"
__ _____
   begin
      A Row := Row Indices2'FIRST;
      M Row := Matrix'FIRST(1);
      Row Loop:
         Toop
            A Col := Col Indices2'FIRST;
            M Col := Matrix'FIRST(2);
            Col Loop:
                loop
```

Answer(A_Row, A_Col) := Matrix(M_Row, M_Col) * Multiplier;



```
exit Col_Loop when M_Col = Matrix'LAST(2);
               A_Col := Col_Indices2'SUCC(A Col);
               M Col := Col Indices1'SUCC(M Col);
            end loop Col Loop;
         exit Row Loop when M Row = Matrix'LAST(1);
         A Row := Row Indices 2'SUCC(A Row);
         M Row := Row Indices1'SUCC(M Row);
      end loop Row_Loop;
   return Answer:
end "*";
```

3.3.6.2.9.8.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:



The following table summarizes the generic types required by this part and defined at the package specification level of Matrix Scalar Operations:

Name	Туре	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col Indices1	discrete type	Used to dimension second dimension of Matrices1
Row_ Indices1	discrete type	Used to dimension first dimension of Matrices1
Col Indices2	discrete type	Used to dimension second dimension of Matrices2
Row_ Indices2	discrete type	Used to dimension first dimension of Matrices2
Matricesl	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:



The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix Scalar Operations Unconstrained package.

Name	Type	Description	
11 11 11	function	Function to define the operation Elements1 * Scalars := Elements2	

3.3.6.2.9.8.10.1.8 LIMITATIONS

None.

3.3.6.2.9.8.10.2 "/" UNIT DESIGN (CATALOG #P427-0)

This function calculates a scaled matrix by dividing each element of an input matrix by a scale factor.

3.3.6.2.9.8.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R074.

3.3.6.2.9.8.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.8.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	1	Туре	1	Mode	Ī	Description
	Matrix Divisor		Matrices2 Scalars		In In		Matrix to be scaled Scale factor

3.3.6.2.9.8.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Col_Loop:

```
Name | Type | Description
| Answer | Matrices1 | Scaled matrix
| A_Col | Col_Indices1 | Index into second dimension of answer matrix
A_Row | Row_Indices1 | Index into first dimension of answer matrix | M_Col | Col_Indices2 | Index into second dimension of input matrix | M_Row | Row_Indices2 | Index into first dimension of input matrix
3.3.6.2.9.8.10.2.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.8.10.2.6 PROCESSING
The following describes the processing performed by this part:
   function "/" (Matrix : Matrices2;
                    Divisor : Scalars) return Matrices1 is
      --declaration section-
      Answer : Matrices1
                     (Row Indices1'FIRST ...
                      Row Indices1'VAL(Matrix'LENGTH(1)-1 +
                                          Row Indices1'POS(Row Indices1'FIRST) ),
                      Col Indices1'FIRST ..
                      Col Indices1'VAL(Matrix'LENGTH(2)-1 +
                                          Col Indices1'POS(Col Indices1'FIRST) ));
      A Col : Col Indices1;
      A Row : Row Indices1;
M Col : Col Indices2;
      M Row : Row Indices2;
-- --begin function "/"
   begin
      A Row := Row Indices1'FIRST;
      M Row := Matrix'FIRST(1);
      Row Loop:
          Гоор
              A Col := Col Indices1'FIRST;
              M Col := Matrix'FIRST(2);
```

Answer(A_Row, A_Col) := Matrix(M_Row, M_Col) / Divisor;

```
exit Col_Loop when M_Col = Matrix'LAST(2);
    A_Col := Col_IndicesI'SUCC(A_Col);
    M_Col := Col_Indices2'SUCC(M_Col);

end loop Col_Loop;

exit Row_Loop when M_Row = Matrix'LAST(1);
    A_Row := Row_IndicesI'SUCC(A_Row);
    M_Row := Row_Indices2'SUCC(M_Row);

end loop Row_Loop;

return Answer;

end "/";
```

3.3.6.2.9.8.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Matrix Scalar Operations:

Name	Туре	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col Indices1	discrete type	Used to dimension second dimension of Matrices1
Row Indices1	discrete type	Used to dimension first dimension of Matrices1
Col Indices2	discrete type	Used to dimension second dimension of Hatrices2
Row Indices2	discrete type	Used to dimension first dimension of Matrices2
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix_Scalar_Operations_Unconstrained package.



Name	Type	Description	Ī
"*"	function	Function to define the operation Elements1 * Scalars := Elements2	1
"/"	function	Function to define the operation Elements2 / Scalars := Elements1	İ

3.3.6.2.9.8.10.2.8 LIMITATIONS

None.

3.3.6.2.9.9 DIAGONAL_MATRIX_SCALAR_OPERATIONS PACKAGE DESIGN (CATALOG #P431-0)

This package provides the functions to allow the user to multiply or divide each element of a diagonal matrix by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.9.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statements which are executed when it is elaborated. This code checks to ensure a square matrix has been instantiated. If not, a Dimension Error exception is raised.

3.3.6.2.9.9.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification:

Data types:

The following table describes the generic formal types required by this part:



Name	Type	Description
Elements1	floating point type	Type of elements in Diagonal_Matrices1
Elements2	floating point type	Type of elements in Diagonal_Matrices2
Scalars	floating point type	Data type of scale factor
Diagonal Rangel	integer type	Used to dimension Diagonal Matrices1
Diagonal Range2	integer type	Used to dimension Diagonal Matrices2
Diagonal Matrices1	array	An array of Elements1
Diagonal_Matrices2	array	An array of Elements2

Subprograms:

The following table describes the generic formal subroutines required by this part:

Ī	Name	1	Туре	1	Description	1
-	11 11		function		Multiplication operator defining the operation: Elements1 * Scalars = Elements2	
İ	m/m	İ	function	İ	Division operator defining the operation: Elements2 / Scalars = Elements1	

3.3.6.2.9.9.4 LOCAL DATA

None.

3.3.6.2.9.9.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.9.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Diagonal_Matrix_Scalar_Operations is

--begin processing for package body

begin

--make sure instantiated diagonal matrices are of the same size
if Diagonal Matrices1'LENGTH /= Diagonal Matrices2'LENGTH then
 raise Dimension Error;

end if;

end Diagonal Matrix Scalar Operations;

3.3.6.2.9.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The fallowing table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

	Name	I	Description	-
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.9.8 LIMITATIONS

The following exceptions are raised by this part:

1	Name		When/Why	Raised	ī
	Dimension_Error		Raised if are not o	the lengths of the two imported vector types of the same length	

3.3.6.2.9.9.9 LLCSC DESIGN

None.

3.3.6.2.9.9.10 UNIT DESIGN

3.3.6.2.9.9.10.1 "*" UNIT DESIGN (CATALOG #P432-0)

This function multiplies each element of a diagonal input matrix by a scale factor.

3.3.6.2.9.9.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.9.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.9.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description	
Matrix	Diagonal_Matrices1	In	Matrix to be scaled	
Multiplier	Scalars	In	Scale factor	

3.3.6.2.9.9.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name		Туре	1	Value		Description
1	Index1	1	Diagonal_Matrices2 Diagonal_Range1 Diagonal_Range2		N/A N/A N/A	İ	Scaled diagonal matrix Index into input matrix Index into output matrix

3.3.6.2.9.9.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.9.10.1.6 PROCESSING

The following describes the processing performed by this part:

-- --declaration section-

Answer : Diagonal_Matrices2;
Index1 : Diagonal_Range1;
Index2 : Diagonal_Range2;

-- --begin function "*"-



```
Index1 := Diagonal_Range1'FIRST;
Index2 := Diagonal_Range2'FIRST;
Process:
    loop

    Answer(Index2) := Matrix(Index1) * Multiplier;
    exit Process when Index1 = Diagonal_Range1'LAST;
    Index1 := Diagonal_Range1'SUCC(Index1);
    Index2 := Diagonal_Range2'SUCC(Index2);
    end loop Process;
    return Answer;
end "*";
```

3.3.6.2.9.9.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined in the package specification of Diagonal Matrix Scalar Operations:

Name	Туре	Description
Elements1	floating point type	Type of elements in Diagonal_Matrices1
Elements2	floating point type	Type of elements in Diagonal_Matrices2
Scalars	floating point type	Data type of scale factor
Diagonal Range1	integer type	Used to dimension Diagonal Matrices1
Diagonal Range2	integer type	Used to dimension Diagonal Matrices2
Diagonal Matrices1	array	An array of Elements1
Diagonal Matrices2	array	An array of Elements2

3.3.6.2.9.9.10.1.8 LIMITATIONS

None.



3.3.6.2.9.9.10.2 "/" UNIT DESIGN (CATALOG #P433-0)

This function divides each element of a diagonal input matrix by a scale factor.

3.3.6.2.9.9.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.9.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.9.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	I	Туре		Mode	Ī	Description	
	Matrix Divisor		Diagonal_Matrices2 Scalars		In In		Matrix to be scaled Scale factor	

3.3.6.2.9.9.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Name	Туре	Value	Description
Ai	nswer	Diagonal_Matrices1	N/A	Scaled diagonal matrix
Ii	ndex1	Diagonal_Range1	N/A	Index into input matrix
Ii	ndex2	Diagonal_Range2	N/A	Index into output matrix

3.3.6.2.9.9.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.9.10.2.6 PROCESSING

The following describes the processing performed by this part:

function "/" (Matrix : Diagonal Matrices2;

Divisor : Scalars) return Diagonal Matrices1 is

```
--declaration section-
     -----
     Answer : Diagonal Matrices1;
     Index1 : Diagonal Rangel;
     Index2 : Diagonal Range2;
-- --begin function "/"-
-- ------
  begin
     Index1 := Diagonal Range1'FIRST;
     Index2 := Diagonal Range2'FIRST;
     Process:
        loop
           Answer(Index1) := Matrix(Index2) / Divisor;
           exit Process when Index1 = Diagonal Rangel'LAST;
           Index1 := Diagonal Range1'SUCC(Index1);
           Index2 := Diagonal Range2'SUCC(Index2);
        end loop Process;
     return Answer;
  end "/";
```

3.3.6.2.9.9.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined in the package specification of Diagonal_Matrix_Scalar Operations:



Name	Type	Description
Elements1	floating point type	Type of elements in Diagonal_Matrices1
Elements2	floating point type	Type of elements in Diagonal_Matrices2
Scalars	floating point type	Data type of scale factor
Diagonal Rangel	integer type	Used to dimension Diagonal Matrices1
Diagonal Range2	integer type	Used to dimension Diagonal Matrices2
Diagonal Matrices1	array	An array of Elements1
Diagonal_Matrices2	array	An array of Elements2

3.3.6.2.9.9.10.2.8 LIMITATIONS

None.

3.3.6.2.9.10 MATRIX MATRIX MULTIPLY UNRESTRICTED PACKAGE DESIGN (CATALOG #P439-0)

This package contains a function which multiplies an m x n matrix by an n x p matrix, returning an m x p matrix. The inner dimensions of the input matrices must be equal, the first dimensions of the left and result matrices must be the same, and the second dimensions of the right and result matrices must be the same. If any of these dimensions do not match, a Dimension Error exception is raised.

The result of this operation is defined as follows:

$$a(i,j) := b(i,k) * c(k,j)$$

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.3.6.2.9.10.2 LOCAL ENTITIES DESIGN

Subprograms:

This package body contains a sequence of statements which are executed when it is elaborated. This code ensures that the dimensions of the instantiated matrices are as required by this part. If they are not, a Dimension_Error exception is raised.

3.3.6.2.9.10.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level of the Matrix Multiply Unrestricted package:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Data type of elements in left input
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
Left_Col_Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left Matrices	array	Data type of left input matrix
Right Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part. To tailor this function to handle sparse matrices, the formal subroutines should be set up to check the appropriate element(s) for zero before performing the indicated operation.

Ī	Name		Туре		Description	<u> </u>
	n+n		function function	į	Function defining the operation Left Elements * Right Elements := Output Elements Function defining the operation	
					Output_Elements + Output_Elements := Output_Elements	



--"p's"

```
3.3.6.2.9.10.4 LOCAL DATA
```

None.

3.3.6.2.9.10.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.10.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General_Vector_Matrix_Algebra)
package body Matrix_Matrix_MultIply_Unrestricted is
```

```
--begin processing for package body
```

begin

```
-- --make sure dimensions are compatible; to be compatible the following
-- --conditions must exist:
-- --must be trying to multiply: [m x n] x [n x p] := [m x p]
if NOT (Left_Matrices'LENGTH(2) = Right_Matrices'LENGTH(1) and --"n's"
Left_Matrices'LENGTH(1) = Output_Matrices'LENGTH(1) and --"m's"
```

Right Matrices'LENGTH(2) = Output Matrices'LENGTH(2)) then

```
-- --dimensions are incompatible
raise Dimension_Error;
```

end if;

end Matrix_Matrix_Multiply_Unrestricted;

3.3.6.2.9.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:



Ī	Name		Description	
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.10.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	I	When/Why Ra:	aised	-
	Dimension_Error		Raised if the [m x n] x [1	he dimensions of the matrices are other than: [n x p] := [m x p]	

3.3.6.2.9.10.9 LLCSC DESIGN

None.

3.3.6.2.9.10.10 UNIT DESIGN

3.3.6.2.9.10.10.1 "*" UNIT DESIGN (CATALOG #P440-0)

This function multiplies an $m \times n$ matrix by an $n \times p$ matrix, returning an $m \times p$ matrix.

The result of this operation is defined as follows:

$$a(i,j) := b(i,k) * c(k,j)$$

3.3.6.2.9.10.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.3.6.2.9.10.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.10.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:



Name	Type	Mode	Description	
Left	Left_Matrices	In	m x n matrix	
Right	Right_Matrices	In	n x p matrix	

3.3.6.2.9.10.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Туре	Value	Description
Answer	Output Matrices	N/A	Result matrix
M_Answer	Output_Row_Indices	N/A	Index into first dimension of result matrix
M_Left	Left_Row_Indices	N/A	Index into first dimension of left input matrix
N_Left	Left_Col_Indices	N/A	Index into second dimension of left input matrix
N_Right	Right_Row_Indices	N/A	Index into first dimension of right input matrix
P_Answer	Output_Col_Indices	N/A	Index into second dimension of result matrix
P_Right	Right_Col_Indices	N/A	Index into second dimension of right input matrix

3.3.6.2.9.10.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.10.10.1.6 PROCESSING

The following describes the processing performed by this part:

function "*" (Left : Left Matrices;

Right : Right Matrices) return Output Matrices is

-- --declaration section-

Answer : Output_Matrices;
M_Answer : Output_Row_Indices;
M_Left : Left_Row_Indices;
N_Left : Left_Col_Indices;
N_Right : Right_Row_Indices;
P_Answer : Output_Col_Indices;
P_Right : Right_Col_Indices;

```
-- --begin of function "*"
  begin
     M Answer := Output Row Indices'FIRST;
              := Left Row Indices'FIRST;
     M_Loop:
        loop
            P Answer := Output Col Indices'FIRST;
            P_Right := Right_Col_Indices'FIRST;
           P_Loop:
              loop
                  Answer(M Answer, P Answer) := 0.0;
                                             := Left Col Indices'FIRST;
                  N Right
                                             := Right_Row_Indices'FIRST;
                 N_Loop:
                     loop
                        Answer(M Answer, P Answer) :=
                           Answer(M Answer, P Answer) +
                           Left(M_Left, N_Left) * Right(N_Right, P Right);
                        exit N Loop when N Left = Left Col Indices'LAST;
                        N Left := Left Col Indices'SUCC(N Left);
                        N_Right := Right_Row_Indices'SUCC(N_Right);
                     end loop N_Loop;
                  exit P_Loop when P_Right = Right_Col_Indices'LAST;
                 P_Right := Right_Col_Indices'SUCC(P_Right);
                 P Answer := Output Col Indices'SUCC(P Answer);
              end loop P Loop;
              exit M_Loop when M_Left = Left_Row_Indices'LAST;
              M Left := Left Row Indices'SUCC(M Left);
              M_Answer := Output_Row_Indices'SUCC(M Answer);
        end loop M_Loop;
     return Answer;
  end "*";
```

3.3.6.2.9.10.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:



Data types:

The following table summarizes the generic types required by this part and defined in the package specification of Matrix_Matrix_Multiply_Unrestricted:

Name	Туре	Description
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
Left_Col_Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices	array	Data type of left input matrix
Right_Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms and task entries:

The following table describes the generic formal subprograms required by this part and defined at the package specification level of Matrix_Matrix_Multiply_-Unrestricted:

Na	me	J T	уре	I	Description	- -
"*"	4	fu	nction	İ	Function defining the operation Left_Elements * Right_Elements := Output_Elements	
"+"		fu	nction		Function defining the operation Output Elements + Output Elements := Output Elements	

3.3.6.2.9.10.10.1.8 LIMITATIONS

None.



3.3.6.2.9.11 MATRIX VECTOR MULTIPLY UNRESTRICTED PACKAGE DESIGN (CATALOG #P434-0)

This package contains a function which multiplies an m x n matrix by an n x 1 vector producing an m x 1 vector. A DIMENSION_ERROR exception is raised if the length of the second dimension of the input matrix is not the same as the length of the input vector or if the length of the first dimension of the length of the same as the length of the output vector.

The result of this operation is defined as follows:

$$a(i) := b(i,j) * c(j)$$

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.11.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO69.

3.3.6.2.9.11.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statements which are executed when the package is elaborated. This section checks the dimensions of the instantiated arrays to ensure they are compatible for a matrix * vector := vector operation. To be compatible the following conditions must exist: Input Matrices: m x n array Input Vectors: n x 1 array Output Vectors: m x 1 array If the dimensions are not compatible, a Dimension Error exception is raised.

3.3.6.2.9.11.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level of Matrix Vector Multiply Unrestricted package:

Data types:

The following table describes the generic formal types required by this part:



Name	Туре	Description
matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input_Vector_Indices	discrete type	Used to dimension input vector
Output_Vector_Indices	discrete type	Used to dimension output vector
Input Matrices	array	Data type of input matrix
Input Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
"*"	function	Function defining the operation
"+" 	function	Function defining the operation Output Vector Elements + Output Vector Elements := Output Vector Elements

3.3.6.2.9.11.4 LOCAL DATA

None.

3.3.6.2.9.11.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.11.6 PROCESSING

The following describes the processing performed by this part:

separate (General Vector Matrix Algebra)
package body Matrix Vector Multiply Unrestricted is

--begin processing for package body

begin

- -- --make sure dimensions are compatible; for dimensions to be compatible the following
 -- --conditions must is what should be requested: [m x n] x [n x 1] = [m x 1]
 if NOT (Input_Matrices'LENGTH(2) = Input_Vectors'LENGTH and --"n's"
 Input_Matrices'LENGTH(1) = Output_Vectors'LENGTH) then --"m's"
- -- --dimensions are incompatible raise Dimension_Error;

end if;

end Matrix_Vector_Multiply_Unrestricted;

3.3.6.2.9.11.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

Ī	Name	1	Description	Ī
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.11.8 LIMITATIONS

The following table describes the exceptions raised by this part:



-	Name		When/Why Raised	Ī
	Dimension_Error		Raised if the length if an operation other than the following is attempted: $[m \times n] \times [n \times 1] := [m \times 1]$	

3.3.6.2.9.11.9 LLCSC DESIGN

None.

3.3.6.2.9.11.10 UNIT DESIGN

3.3.6.2.9.11.10.1 "*" UNIT DESIGN (CATALOG #P435-0)

This function multiplies an m \times n matrix by an n \times 1 vector producing an m \times 1 vector.

The result of this operation is defined as follows:

$$a(i) := b(i,j) * c(j)$$

3.3.6.2.9.11.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO69.

3.3.6.2.9.11.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.11.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type		Mode		Description	-
Matrix Vector	Input_Matrices Input_Vectors		In In		Matrix to be used as the multiplicand Vector to be used as the multiplier	

3.3.6.2.9.11.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:



1	Name	Туре	Description
1	Answer	Output_Vectors	Result of performing the matrix-vector multiplication
	M_Answer M_Matrix N_Matrix	Output_Vector_Indices Row_Indices Col Indices	Index into result vector Index into input matrix Index into input matrix
	N_Vector	Input_Vector_Indices	Index into input waterx

3.3.6.2.9.11.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.11.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function "*" (Matrix: Input Matrices;

Vector: Input Vectors) return Output Vectors is
```

```
Answer : Output_Vectors;
M_Answer : Output_Vector_Indices;
M_Matrix : Row_Indices;
N_Matrix : Col_Indices;
```

--declaration section-

N_Vector : Input_Vector_Indices;

```
-- --begin function "*"
```

begin

```
M_Matrix := Row_Indices'FIRST;
M_Loop:
    loop

Answer(M_Answer) := 0.0;
    N_Matrix := Col_Indices'FIRST;
    N_Vector := Input_Vector_Indices'FIRST;
    N_Loop:
    loop
```

M Answer := Output Vector Indices'FIRST;

```
exit N Loop when N Matrix = Col Indices'LAST;
N Matrix := Col Indices'SUCC(N Matrix);
N_Vector := Input_Vector_Indices'SUCC(N_Vector);
```

```
end loop N_Loop;

exit M_Loop when M_Matrix = Row_Indices'LAST;
    M_Matrix := Row_Indices'SUCC(M_Matrix);
    M_Answer := Output_Vector_Indices'SUCC(M_Answer);

end loop M_Loop;

return Answer;
end "*";
```

3.3.6.2.9.11.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table describes the generic data types required by this part and defined at the package specification level of Matrix_Vector_Multiply_Unrestricted package:

Name	Туре	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input_Vector_Indices	discrete type	Used to dimension input vector
Output_Vector_Indices	discrete type	Used to dimension output vector
Input Matrices	array	Data type of input matrix
Input Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms and task entries:

The following table summarizes the generic formal subroutines and required by this part and defined at the package specification level of Matrix_Vector_- Multiply Unrestricted:



Name	Type	Description
"*" 	function	Function defining the operation Matrix_Elements * Input_Vector_Elements := Output_Vector_Elements
"+" 	function	Function defining the operation Output Vector Elements + Output Vector Elements := Output_Vector Elements

3.3.6.2.9.11.10.1.8 LIMITATIONS

None.

3.3.6-2.9.12 VECTOR_VECTOR_TRANSPOSE_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P442-0)

This function multiplies one input vector by the transpose of a second input vector, returning the resultant matrix. This package expects the instantiated arrays to have the following dimensions:

Left_Vectors : m x 1 array Right_Vectors : n x 1 array Matrices : m x n array

If the dimensions are not as expected, a Dimension Error exception is raised.

The following defines the result of this operation:

$$a(i,j) := b(i) * c(j)$$

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.12.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.12.2 LOCAL ENTITIES DESIGN

Subprograms:

This package body contains a sequence of statements which checks ensure the dimensions of the instantiated vectors and arrays are required by this part. If they are not, a Dimension Error exception is raised.

3.3.6.2.9.12.3 INPUT/OUTPUT

GENERIC PARAMETERS:



The following generic parameters were previously defined at the package specification level of the Vector_Vector_Transpose_Multiply_Unrestricted package:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Left_Vector_Indices	discrete type	Used to dimension left input vector
Right_Vector_Indices	discrete type	Used to dimension right input vector
Col_Indices	discrete type	Used to dimension second dimension of output matrix
Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left Vectors	array	Data type of left input vector
Right Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

Subprograms:

The following table describes the generic formal subroutines required by this part:

Ī	Name		Туре	1	Description	. <u> </u>
	11 🛧 11		function		Operator defining the multiplication operation Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements	

3.3.6.2.9.12.4 LOCAL DATA

None.

3.3.6.2.9.12.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.12.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Vector Vector Transpose Multiply_Unrestricted is

--begin processing for package body

begin

-- -- make sure dimensions are compatible; must have the following conditions:

-- -- attempted operation is [m x 1] x [1 x n] := [m x n]
if NOT (Left_Vectors'LENGTH = Matrices'LENGTH(1) and -- "m's"
Right_Vectors'LENGTH = Matrices'LENGTH(2)) then -- "n's"

raise Dimension Error;

end if:

end Vector Vector Transpose Multiply_Unrestricted;

3.3.6.2.9.12.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

Name	Description	
dimension	n_error Raised by a routine when input in dimensions incompatible for the operation to be performed	received has e type of

3.3.6.2.9.12.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	1	When/Why Raised	
	Dimension_Error		Raised if an attempt is made to put the result of $[m \times 1]$ vector $\times [1 \times n]$ vector into other than a $[m \times n]$ matrix	

3.3.6.2.9.12.9 LLCSC DESIGN

None.

3.3.6.2.9.12.10 UNIT DESIGN

3.3.6.2.9.12.10.1 "*" UNIT DESIGN (CATALOG #P443-0)

This function multiplies one input vector by the transpose of a second input vector, returning the resultant matrix.

The following defines the result of this operation:

$$a(i,j) := b(i) * c(j)$$

3.3.6.2.9.12.10.1.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.12.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.12.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description	
Left	Left_Vectors	In	m x 1 vector	
Right	Right_Vectors	In	1 x n vector	

3.3.6.2.9.12.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

^^	_
٧,۶	•
V١.	*
-	

Ī	Name	Type	Value	Description
ī	Answer	Matrices	N/A	Result .rix
İ	M_Answer	Row_Indices	N/A	Index into first dimension of output matrix
Ì	M Left	Left Vector Indices	N/A	Index into left input vector
İ	N_Answer	Col_Indices	N/A	Index into second dimension of output matrix
Ì	N_Right	Right_Vector_Indices	N/A	Index into right input vector

3.3.6.2.9.12.10.1.5 PROCESS CONTROL

Not applicable.

M Loop:

3.3.6.2.9.12.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function "*" (Left : Left_Vectors ;
```

Right: Right Vectors) return Matrices is

```
loop

N_Right := Right_Vector_Indices'FIRST;
N_Answer := Col_Indices'FIRST;
N_Loop:
    loop
```

Answer(M_Answer, N_Answer) := Left(M_Left) * Right(N_Right);

```
exit N_Loop when N_Right = Right_Vector_Indices'LAST;
N_Right := Right_Vector_Indices'SUCC(N_Right);
N_Answer := Col_Indices'SUCC(N_Answer);
```



```
end loop N_Loop;

exit M_Loop when M_Answer = Row_Indices'LAST;
    M_Answer := Row_Indices'SUCC(M_Answer);
    M_Left := Left_Vector_Indices'SUCC(M_Left);
end loop M_Loop;
return Answer;
end "*";
```

3.3.6.2.9.12.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of the Vector_Vector_Transpose_-Multiply Unrestricted package:

Name	Туре	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Left_Vector_Indices	discrete type	Used to dimension left input vector
Right_Vector_Indices	discrete type	Used to dimension right input vector
Col_Indices	discrete type	Used to dimension second dimension of output matrix
Row_Indices	discrete type	Used to dimension first dimension of output matrix
Left Vectors	array	Data type of left input vector
Right Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined at the package specification level of the Vector_Vector_-Transpose_Multiply_Unrestricted package:

1	`~	٠,	
	_		

<u></u>	Name	 	Туре		Description	- <u>-</u>
	77 H		function		Operator defining the multiplication operation Left_Vector_Elements * Right_Vector_Elements := Matrix_Elements	1

3.3.6.2.9.12.10.1.8 LIMITATIONS

None.

3.3.6.2.9.13 MATRIX_MATRIX_TRANSPOSE_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P445-0)

This package contains a function which multiplies one input matrix by the transpose of a second input matrix, returning the resultant matrix. The results of this operation are defined as follows:

$$a(i,j) := b(i,k) * c(j,k)$$

This package expects the instantiated arrays to have been dimensioned as follows:

Left_Matrices: m x n matrix Right_Matrices: p x n matrix Output_Matrices: m x p matrix

If the matrices have not been instantiated as expected, a Dimension_Error exception is raised.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.13.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.13.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statements which are executed when it is elaborated. This code checks to ensure the dimensions of the instantiated matrices are as required for this part. If not, a Dimension_Error exception is raised.

3.3.6.2.9.13.3 INPUT/OUTPUT

GENERIC PARAMETERS:



The following generic parameters were previously defined at the package specification level of the Matrix_Matrix_Transpose_Multiply_Unrestricted package:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Туре	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
Left_Col_ Indices	discrete type	Used to dimension second dimension of left input matrix
Left_Row_ Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_ Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_ Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_ Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_ Indices	discrete type	Used to dimension first dimension of output matrix
Left_Matrices Right Matrices	array	Data type of left input matrix Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Ī	Name	1	Туре	ı	Description	Ī
	"*"		function		Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements	

3.3.6.2.9.13.4 LOCAL DATA

None.

3.3.6.2.9.13.5 PROCESS CONTROL

Not applicable.





3.3.6.2.9.13.6 PROCESSING

The following describes the processing performed by this part:

separate (General Vector Matrix Algebra)
package body Matrix Matrix Transpose Multiply Unrestricted is

```
--begin processing for package body
```

raise Dimension Error;

end if;

end Matrix Matrix Transpose Multiply Unrestricted;

3.3.6.2.9.13.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

Name	Description	
Dimension_Error	Raised by a routine when input receive dimensions incompatible for the type operation to be performed	d has of

3.3.6.2.9.13.8 LIMITATIONS

The following table describes the exceptions raised by this part:



(

	Na	ime		 	When/Why	Raised	-
	Dim	ension_I	Error		matrices	the m's, n's, and p's of the input and output are not equal; i.e., need to be doing the g operation: [m x n] x [p x n] := [m x p]	

3.3.6.2.9.13.9 LLCSC DESIGN

None.

3.3.6.2.9.13.10 UNIT DESIGN

3.3.6.2.9.13.10.1 "*" UNIT DESIGN (CATALOG #P446-0)

This function multiples an $m \times n$ matrix by the transpose of a $p \times n$ matrix, returning the resultant $m \times p$ matrix.

The results of this operation are defined as follows:

$$a(i,j) := b(i,k) * c(j,k)$$

3.3.6.2.9.13.10.1.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.13.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.13.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Ī	Mode	1	Description	Ī
L	eft ight	Left_Matrices Right_Matrices		In In		Matrix to be used as the multiplicand Matrix whose transpose is to be used as the multiplier	

3.3.6.2.9.13.10.1.4 LOCAL DATA

Data objects:



The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Output Matrices	N/A	Result matrix being calculated
M_Answer	Output_Row_Indices	N/A	Index into first dimension of result matrix
M_Left	Left_Row_Indices	N/A	Index into first dimension of left input matrix
N_Left	Left_Col_Indices	N/A	Index into second dimension of left input matrix
N_Right	Right_Col_Indices	N/A	Index into second dimension of right input matrix
P_Answer	Output_Col_Indices	N/A	Index into second dimension of result matrix
P_Right	Right_Row_Indices	N/A	Index into first dimension of right input matrix

3.3.6.2.9.13.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.13.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function "*" (Left : Left Matrices;
```

Right: Right Matrices) return Output Matrices is

```
Answer : Output Matrices;
M_Answer : Output Row_Indices;
M_Left : Left Row_Indices;
N_Left : Left Col_Indices;
N_Right : Right_Col_Indices;
P_Answer : Output Col_Indices;
P_Right : Right_Row_Indices;
```

--declaration section

```
begin
```

-- --begin function "*"

```
M_Answer := Output_Row_Indices'FIRST;
M_Left := Left_Row_Indices'FIRST;
M_Loop:
    loop
```

P_Answer := Output_Col_Indices'FIRST;



```
P Right := Right Row Indices'FIRST;
         P Loop:
            loop
               Answer(M Answer, P Answer) := 0.0;
               N Left := Left Col Indices'FIRST:
               N Right := Right Col Indices'FIRST;
               N Loop:
                  loop
                     Answer(M Answer, P Answer) :=
                        Answer(M Answer, P Answer) +
                        Left(M Left, N Left) * Right(P Right, N Right);
                     exit N Loop when N Left = Left Col Indices'LAST;
                     N Left := Left Col Indices'SUCC(N Left);
                     N Right := Right Col Indices' SUCC(N Right);
                  end loop N Loop;
               exit P Loop when P Answer = Output Col Indices'LAST;
               P Answer := Output Col Indices'SUCC(P Answer);
               P Right := Right Row Indices' SUCC(P Right);
            end loop P Loop;
         exit M Loop when M Answer = Output Row Indices'LAST;
        M Answer := Output Row Indices SUCC(M Answer);
        M Left := Left Row Indices'SUCC(M Left);
      end loop M Loop;
  return Answer;
end "*":
```

3.3.6.2.9.13.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification of Matrix_Matrix_Transpose_Multiply_-Unrestricted:



Name	Type	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
Left_Col_ Indices	discrete type	Used to dimension second dimension of left input matrix
Left Row_ Indices	discrete type	Used to dimension first dimension of left input matrix
Right_Col_ Indices	discrete type	Used to dimension second dimension of right input matrix
Right_Row_ Indices	discrete type	Used to dimension first dimension of right input matrix
Output_Col_ Indices	discrete type	Used to dimension second dimension of output matrix
Output_Row_ Indices	discrete type	Used to dimension first dimension of output matrix
Left Matrices Right Matrices	array array	Data type of left input matrix Data type of right input matrix
Output_Matrices	array	Data type of output matrix

Subprograms and task entries:

The following table summarizes the generic formal subroutines and task entries required by this part and defined at the package specification level of the Matrix Transpose_Multiply_Unrestricted package:

Ī	Name	Ī	Type	1	Description	
	n⊁u		function		Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements	

3.3.6.2.9.13.10.1.8 LIMITATIONS

None.

3.3.6.2.9.14 DOT_PRODUCT_OPERATIONS_UNRESTRICTED PACKAGE DESIGN (CATALOG #P448-0)

This package contains a function which performs a dot product operation on two m-element vectors.

The decomposition for this part is the same as that shown in the "no-Level Design Document.



3.3.6.2.9.14.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.3.6.2.9.14.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains a sequence of statement which are executed when the package is elaborated. This code checks to ensure the lengths of the instantiated vectors are the same.

3.3.6.2.9.14.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in this part's package specification:

Data types:

The following table describes the generic formal types required by this part:

Ī	Name	Туре	I	Description
	Left_Elements	floating point type		Type of elements in left input vector
	Right_Elements	floating point type		Type of elements in right input vector
Ì	Result_Elements	floating point type		Data type of result of dot product
İ	Left Indices	discrete	ĺ	Used to dimension Left Vectors
İ	Right Indices	discrete	İ	Used to dimension Right Vectors
İ	Left Vectors	array	İ	Data type of left input vector
İ	Right_Vectors	array	İ	Data type of right input vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

Ī	Name		Туре		Description	Ī
	"*"		function		Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements	







3.3.6.2.9.14.4 LOCAL DATA

None.

3.3.6.2.9.14.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.14.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Dot_Product_Operations_Unrestricted is

--begin processing for package body

begin

-- --make sure instantiated vectors are of the same length
if Left_Vectors'LENGTH /= Right_Vectors'LENGTH then
 raise Dimension_Error;
end if;

end Dot Product Operations Unrestricted;

3.3.6.2.9.14.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in ancestral units:

Name		Description	<u> </u>
Dimension_Error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	



3.3.6.2.9.14.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name	When/Why	Raised	-
Ī	Dimension_Error	Raised if the same	the lengths of the two input vectors is not	1

3.3.6.2.9.14.9 LLCSC DESIGN

None.

3.3.6.2.9.14.10 UNIT DESIGN

3.3.6.2.9.14.10.1 DOT PRODUCT UNIT DESIGN (CATALOG #P449-0)

This function performs a dot product operation on two m-element vectors.

3.3.6.2.9.14.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R063.

3.3.6.2.9.14.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.14.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	 	Type		Mode	1	Description
	Left Right		Left_Vectors Right_Vectors		ţu ru		First /ector in a dot product operation Second vector in a dot product operation

3.3.6.2.9.14.10.1.4 LOCAL DATA

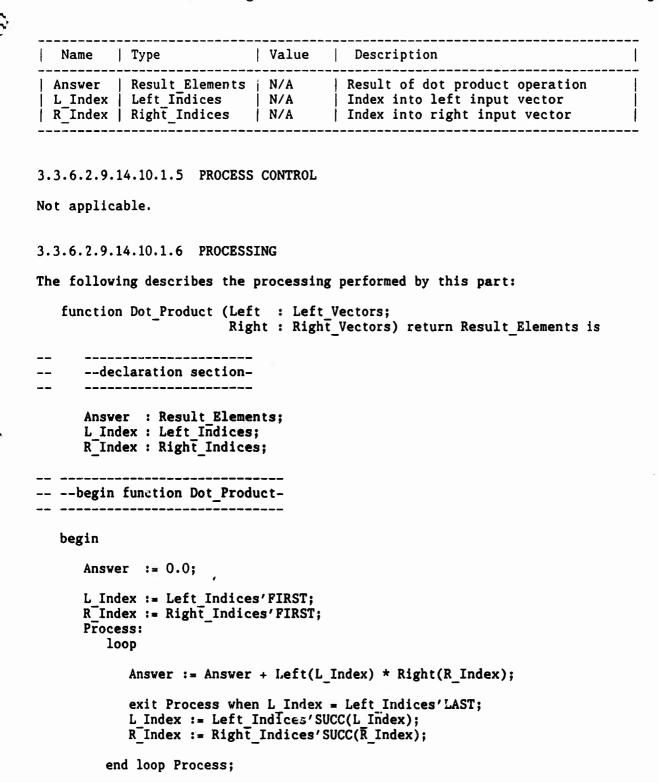
Data objects:

The following table describes the data objects maintained by this part:

The

return Answer;

end Dot Product;



3.3.6.2.9.14.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of the Dot_Product_Operations_-Unrestricted package:

Ī	Name	Type	Description
1	Left_Elements	floating point type	Type of elements in left input vector
İ	Right_Elements	floating point type	Type of elements in right input vector
İ	Result_Elements	floating point type	Data type of result of dot product
ĺ	Left Indices	discrete	Used to dimension Left Vectors
İ	Right Indices	discrete	Used to dimension Right Vectors
İ	Left Vectors	array	Data type of left input vector
İ	Right_Vectors	array	Data type of right input vector

3.3.6.2.9.14.10.1.8 LIMITATIONS

None.

3.3.6.2.9.15 DIAGONAL_FULL_MATRIX_ADD_UNRESTRICTED PACKAGE DESIGN (CATALOG #P451-0)

This package contains a function adds a diagonal matrix to a full matrix by adding the individual elements of the input matrices.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.15.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.15.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains code which is executed when the package is elaborated. This code checks to make sure the dimensions of the instantiated arrays are compatible. The diagonal matrix should have m elements, and both of the full matrices should be m x m arrays. If these conditions are not met, a



Dimension_Error exception is raised.

3.3.6.2.9.15.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification of Diagonal Full Matrix Add Unrestricted:

Data types:

The following table describes the generic formal types required by this part:

Name		Type	Description
Elements		floating point type	Type of elements in input and output arrays
Diagonal	Range	integer type	Used to dimension Diagonal Matrices
Full_Inpu	it_Col_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Inpu	t_Row_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Outp	out_Col_Indices	discrete	Used to dimension Full_Output_ matrices
Full_Outp	out_Row_Indices	discrete	Used to dimension Full_Output_ matrices
Diagonal	Matrices	array	Data type of diagonal input matrix
	it_Matrices	array	Data type of full input matrix
Full_Outp	out_Matrices	array	Data type of full output matrix

3.3.6.2.9.15.4 LOCAL DATA

None.

3.3.6.2.9.15.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.15.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector Matrix_Algebra)
package body Diagonal_Full_Matrix_Add_Unrestricted is





```
begin
```

```
-- --make sure square matrices of the same size have been instantiated if not (Diagonal_Matrices'LENGTH = Full_Input_Matrices'LENGTH(1) and Full_Input_Matrices'LENGTH(1) = Full_Input_Matrices'LENGTH(2) and Full_Input_Matrices'LENGTH(1) = Full_Output_Matrices'LENGTH(1) and Full_Output_Matrices'LENGTH(1) = Full_Output_Matrices'LENCTH(2)) then raise Dimension_Error;
```

end if;

end Diagonal Full Matrix Add Unrestricted;

3.3.6.2.9.15.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General Vector Matrix Algebra:

Ī	Name		Description
	Dimension_Error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.15.8 LIMITATIONS

The following table describes the exceptions raised by this part:

1	Name	1	When/Why	Raised	Ī
	Dimension_Error		Raised if equal to matrix	the lengths of the matrix indices are not each other and other the length of the diagonal	

3.3.6.2.9.15.9 LLCSC DESIGN

None.



3.3.6.2.9.15.10 UNIT DESIGN

3.3.6.2.9.15.10.1 "+" UNIT DESIGN (CATALOG #P452-0)

This function adds an m-element diagonal matrix to an m x m matrix, returning the resultant m x m matrix.

3.3.6.2.9.15.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.9.15.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.15.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре	1	Mode]	Description	-
	D_Matrix F_Matrix		Diagonal_Matrices Full_Input_Matrices		In In	 	Input diagonal matrix Input full matrix to be added to the diagonal matrix	

3.3.6.2.9.15.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:



	-
	-
d dimen- matrix	
n Answer	
ntains ement in	
: dimen-	

Name	Type	Value	Description
Answer	Full Output Matrices	N/A	Resultant matrix
A_Col_Index	Full_Output_Col_Indices	N/A	Index into second dimen- sion of Answer matrix
A_Col_Marker	Full_Output_Col_Indices	N/A 	Marks a column in Answer matrix which contains the diagonal element in row A Row Index
A_Row_Index	Full_Output_Row_Indices	N/A	Index Into first dimen- sion of Answer matrix
D_Index	Diagonal_Range	N/A	Index into diagonal matrix
F_Col_Index	Full_Input_Col_Indices	N/A	Index into second dimen- sion of F Matrix
F_Row_Index	Full_Input_Row_Indices	N/A	Index into first dimen- sion of F_Matrix

3.3.6.2.9.15.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.15.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function "+" (D Matrix : Diagonal_Matrices;
```

F Matrix: Full Input Matrices) return Full Output Matrices is

```
--declaration section-
_____
```

```
: Full Output Matrices;
Answer
A Col Index : Full Output Col Indices;
A_Col_Marker : Full_Output_Col_Indices;
A_Row_Index : Full_Output_Row_Indices;
```

D Index : Diagonal Range;

F Col Index : Full Input Col Indices; F Row Index : Full Input Row Indices;

```
- --begin function "+"
__ _____
```

begin

--first assign a row full of values, then add in diagonal element

```
A_Col_Marker := Full_Output_Col_Indices'FIRST;
A Row Index := Full Output Row Indices' FIRST;
```

D Index := Diagonal Range'FIRST;

F Row Index := Full Input Row Indices'FIRST;





Add_Loop: Toop

```
F Col Index := Full Input Col Indices'FIRST;
           Assign Loop:
              loop
                  Answer(A Row Index, A Col Index) :=
                      F_Matrix(F_Row_Index, F_Col_Index);
                  exit Assign Loop
                      when A Col Index = Full Output Col Indices'LAST;
                  A Col Index := Full Output Col Indices'SUCC(A Col Index);
F_Col_Index := Full_Input_Col_Indices'SUCC(F_Col_Index);
              end loop Assign Loop;
          Answer(A Row Index, A Col Marker) :=
              Answer(A Row Index, A Col Marker) + D Matrix(D Index);
          exit Add Loop when D Index = Diagonal Range'LAST;
          A Col Marker := Full_Output_Col_Indices'SUCC(A Col Marker);
A_Row_Index := Full_Output_Row_Indices'SUCC(A_Row_Index);
          D Index
                          := D Index + 1;
          F Row Index := Full Input Row Indices'SUCC(F Row Index);
       end loop Add Loop;
   return Answer;
end "+":
```

A Col Index := Full Output Col Indices'FIRST;

3.3.6.2.9.15.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Diagonal_Full_Matrix_Add_-Unrestricted:



Name	Туре	Description
Elements	floating point type	Type of elements in input and
Diagonal_Range	integer type	Used to dimension Diagonal_Matrices
Full_Input_Col_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Input_Row_Indices	discrete	Used to dimension Full_Input_ matrices
Full_Output_Col_Indices	discrete	Used to dimension Full_Output_ matrices
Full_Output_Row_Indices	discrete	Used to dimension Full_Output_ matrices
Diagonal Matrices	array	Data type of diagonal input matrix
Full_Input_Matrices	array	Data type of full input matrix
Full_Output_Matrices	array	Data type of full output matrix

3.3.6.2.9.15.10.1.8 LIMITATIONS

None.

3.3.6.2.9.16 VECTOR_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P342-0)

This package contains functions which provide a set of standard vector operations. The operations provided are addition, subtraction, and dot product of like vectors, along with a vector length operation.

The vectors operated upon by parts in this part are constrained arrays.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.16.1 REQUIREMENTS ALLOCATION

The following table describes the allowing of requirements to this part:

	Requirements
Name	Allocation
Dot_Product Vector_Length "+"	R063 R104 R061 R062







3.3.6.2.9.16.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were defined at the package specification level:

Data types:

Name	Туре	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector Elements * Vector Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

Subprograms:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements Squared
SqRt	function	Square root function taking an object of type Vector Elements Squared and returning an object of type Vector Elements

3.3.6.2.9.16.4 LOCAL DATA

Data types:

The following table summarizes the types defined in this part's specification:

 	Name	Range	Description	
	Vectors	N/A	Constrained, one-dimensional array of elements	



3.3.6.2.9.16.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Vector_Operations_Constrained is

end Vector Operations Constrained;

3.3.6.2.9.16.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.16.8 LIMITATIONS

None.

3.3.6.2.9.16.9 LLCSC DESIGN

None.

3.3.6.2.9.16.10 UNIT DESIGN

3.3.6.2.9.16.10.1 "+" (VECTOR + VECTORS) = VECTORS) UNIT DESIGN (CATALOG #P343-0)

This function adds two vectors by adding each of the individual elements in the input vector, returning the resultant vector.

3.3.6.2.9.16.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO61.

3.3.6.2.9.16.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:



Ī	Name	 	Туре	Mode	Description
	Left Right		Vectors Vectors	In In	One of the vectors to be added Second vector to be added

3.3.6.2.9.16.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name		Description	
•	•	Vector being calculated and returned	

3.3.6.2.9.16.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.1.6 PROCESSING

The following describes the processing performed by this part:

begin

-- --begin function "+"

```
Process:
    for Index in Indices loop
        Answer(Index) := Left(Index) + Right(Index);
    end loop Process;
    return Answer;
end "+";
```



3.3.6.2.9.16.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and are defined in the package specification for Vector Operations Constrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector Operations Constrained:

Ī	Name	Range	Description
	Vectors	N/A	Constrained, one-dimensional array of elements

3.3.6.2.9.16.10.1.8 LIMITATIONS

None.

3.3.6.2.9.16.10.2 "-" (VECTORS - VECTORS := VECTORS) UNIT DESIGN (CATALOG #P344-0)

This part subtracts one vector from another by subtracting the individual elements of each input vector, returning the resultant vector.

3.3.6.2.9.16.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO62.

3.3.6.2.9.16.10.2.2 LOCAL ENTITIES DESIGN

None.





3.3.6.2.9.16.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description	
Left	Vectors	In	Vector to act as the minuend	
Right	Vectors	In	Vector to act as the subtrahend	

3.3.6.2.9.16.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	,	Туре	 	Description]
Answe	r	Vectors	V	ector being calculated and returned	

3.3.6.2.9.16.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.2.6 PROCESSING

The following describes the processing performed by this part:

Process:
for Index in Indices loop

Answer(Index) := Left(Index) - Right(Index);
end loop Process;



return Answer;

end "-":

3.3.6.2.9.16.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector_Operations_Constrained:

Name	Type	Description
Vector_Elements Indices	floating point type discrete type	Type of elements to be contained in vector type defined by this package Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Constrained:

Ī	Name	Ī	Range		Description	1
	Vectors		N/A		Constrained, one-dimensional array of elements	

3.3.6.2.9.16.10.2.8 LIMITATIONS

None.

3.3.6.2.9.16.10.3 VECTOR LENGTH UNIT DESIGN (CATALOG #P345-0)

This function calculates the length of a vector, returning the result. The length of a vector is defined as:

a := Sqrt(sum b(i)**2)

3.3.6.2.9.16.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R104.



3.3.6.2.9.16.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

•	Type	I	Mode	1		pti	on					
Input	Vectors	1	In	1	Vector	for	which	а	length	is	desired	•

3.3.6.2.9.16.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Туре	Description	I
Temp	Vector_Elements_S	Squared Used for intermediate calculation	ns

3.3.6.2.9.16.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.3.6 PROCESSING

The following describes the processing performed by this part:

function Vector Length (Input: Vectors) return Vector Elements is

```
-- --declaration section-
```

Temp : Vector Elements Squared;

```
-- --begin function Vector_Length
```

begin

Temp := 0.0;

Process:



3.3.6.2.9.16.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector Operations Constrained:

Name	Туре	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector_Elements_ Squared	floating point type	Resulting type from the operation Vector Elements * Vector Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Constrained:

Name	Range	Description	
Vectors	N/A	Constrained, one-dimensional array of elements	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Description	- -
dimension_error	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	



Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level for Vector_Operations_- Constrained:

Name	Type	Description
"*"	function	Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements Squared
SqRt	function	Square root function taking an object of type Vector Elements Squared and returning an object of type Vector Elements

3.3.6.2.9.16.10.3.8 LIMITATIONS

None.

3.3.6.2.9.16.10.4 DOT PRODUCT UNIT DESIGN (CATALOG #P346-0)

This function calculates the dot product of two vectors by keeping a running sum of the product of each element of the input vectors.

3.3.6.2.9.16.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO63.

3.3.6.2.9.16.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.16.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	I	lype	ı	Mode	I	Description
	Left		Vectors		In		First vector to be used in the dot product operation
İ	Right	İ	Vectors		In	İ	Second vector to be used in the dot product operation



```
3.3.6.2.9.16.10.4.4 LOCAL DATA
```

Data objects:

The following table describes the data objects maintained by this part:

Name Type	Value	Description
Answer Vector_Elements_Squared	N/A 	Result of dot product operation

3.3.6.2.9.16.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.16.10.4.6 PROCESSING

```
The following describes the processing performed by this part:
```

```
function Dot_Product (Left : Vectors;
Right : Vectors) return Vector_Elements_Squared is
```

-- --declaration section

Answer: Vector Elements Squared;

```
-- --begin function Dot_Product
```

begin

```
Answer := 0.0;
Process:
    for Index in Indices loop
        Answer := Answer + Left(Index) * Right(Index);
    end loop Process;
    return Answer;
end Dot_Product;
```

3.3.6.2.9.16.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:



The following tables describe the elements defined in this part's top level component and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level for Vector Operations Constrained:

Name	Type	Description
Vector_Elements	floating point type	Type of elements to be contained in vector type defined by this package
Vector Elements	floating	Resulting type from the operation
Squared	point type	Vector Elements * Vector Elements; used for result of a dot product operation
Indices	discrete type	Used to dimension exported Vectors type

The following table summarizes the types required by this part and defined in the package specification for Vector_Operations_Constrained:

Name	I	Range		Description
Vectors		N/A		Constrained, one-dimensional array of elements

Subprograms:

The following table summarizes the generic subroutines available to this part and defined at the package specification level for Vector Operations:

Name		Туре		Description	Ī
#±#		function		Used to define the operation Vector_Elements * Vector_Elements := Vector_Elements_Squared	

3.3.6.2.9.16.10.4.8 LIMITATIONS

None.

3.3.6.2.9.17 MATRIX_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P355-0)

This package contains subroutines which provide a set of standard operations on matrices of like types.

The decomposition for this part is the same as that shown in the Top-Level Design Document.



3.3.6.2.9.17.1 REQUIREMENTS ALLOCATION

This following illustrates the allocation of requirements to the units in this package.

 Name	Requirements Allocation	
"+" (matrices + matrices) "-" (matrices - matrices) "+" (matrices + elements) "-" (matrices - elements) Set_to_Identity_Matrix Set_to_Zero_Matrix	R079 R080 R075 R076 R155 R156	

3.3.6.2.9.17.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Name	Type	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

3.3.6.2.9.17.4 LOCAL DATA

Data types:

The following data type was previously defined at the package specification level:

Ī	Name	Range	Description	- -
	Matrices	N/A 	Constrained, two-dimensional array of Elements	- -



3.3.6.2.9.17.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Matrix_Operations_Constrained is

end Matrix_Operations_Constrained;

3.3.6.2.9.17.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.17.8 LIMITATIONS

None.

3.3.6.2.9.17.9 LLCSC DESIGN

None.

3.3.6.2.9.17.10 UNIT DESIGN

3.3.6.2.9.17.10.1 "+" (MATRICES + MATRICES := MATRICES) UNIT DESIGN (CATALOG #P356-0)

This function adds two matrices by adding the individual elements of each input matrix, returning the resultant matrix.

3.3.6.2.9.17.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R079.

3.3.6.2.9.17.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:



Ī	Name		Туре	ı	Mode		Description
	Left Right		Matrices Matrices		In In		First matrix to be added Second matrix to be added

3.3.6.2.9.17.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Туре	Description	
•	Matrices	Result of adding the two input matrices	

3.3.6.2.9.17.10.1.5 PROCESS CONTROL

end loop Row Loop;

Not applicable.

3.3.6.2.9.17.10.1.6 PROCESSING

The following describes the processing performed by this part:

begin



return Answer:

end "+";

3.3.6.2.9.17.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Ī	Name	Type	Description	1
- 	Elements	floating point type	Used to define type of elements in matrix defined by this package	
j	Col_Indices	discrete type	Used to define second dimension of exported matrix type	İ
	Row_Indices	discrete type	Used to define first dimension of exported matrix type	

The following table summarizes the types required by this part and defined in the package specification of Matrix Operations Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.1.8 LIMITATIONS

None.

3.3.6.2.9.17.10.2 "-" (MATRICES - MATRICES := MATRICES) UNIT DESIGN (CATALOG #P357-0)

This function subtracts one matrix from another by subtracting the individual elements of the input matrices, returning the resultant matrix.

3.3.6.2.9.17.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO80.



3.3.6.2.9.17.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	Matrix to act as the minuend
Right	Matrices	In	Matrix to be used as the subtrahend

3.3.6.2.9.17.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description	1
Answer	Matrices	Result of adding the two input matrices	Ī

3.3.6.2.9.17.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.2.6 PROCESSING

The following describes the processing performed by this part:

function "-" (Left : Matrices; Right : Matrices) return Matrices is

-- --declaration section-

Answer : Matrices;

-- --begin function "-"

begin

Row_Loop:



3.3.6.2.9.17.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:



The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Туре	Description	
Elements	floating point type	Used to define type of elements in matrix defined by this package	
Col_Indices	discrete type	Used to define second dimension of exported matrix type	İ
Row_Indices	discrete type	Used to define first dimension of exported matrix type	

The following table summarizes the types required by this part and defined in the package specification for Matrix_Operations_Constrained:

Ī	Name	Ī	Range		Description
	Matrices		N/A		Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.2.8 LIMITATIONS

None.



3.3.6.2.9.17.10.3 "+" (MATRICES + ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P358-0)

This function calculates a scaled matrix by adding a scale factor to each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.17.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R075.

3.3.6.2.9.17.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	Type	Mode	Description	
-	Matrix Addend	Matrices Elements	In In	Matrix to be scaled Scale factor	-

3.3.6.2.9.17.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name Type		 -	Description	-
Ī	Answer			Scaled matrix	

3.3.6.2.9.17.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.3.6 PROCESSING

The following describes the processing performed by this part:

function "+" (Matrix : Matrices;

Addend : Elements) return Matrices is

-- --------



3.3.6.2.9.17.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix Operations Constrained:

1	Name	Туре	Description	
	Elements	floating point type	Used to define type of elements in matrix defined by this package	
	Col_Indices	discrete type	Used to define second dimension of exported matrix type	İ
	Row_Indices	discrete type	Used to define first dimension of exported matrix type	

The following table summarizes the types required by this part and defined in the package specification of Matrix Operations Constrained:

Name	Range	Description	!
Matrices	N/A	Constrained, two-dimensional array of Elements	



3.3.6.2.9.17.10.3.8 LIMITATIONS

None.

3.3.6.2.9.17.10.4 "-" (MATRICES - ELEMENTS := MATRICES) UNIT DESIGN (CATALOG #P359-0)

This function calculates a scaled matrix by subtracting a scale factor from each element of an input matrix, returning the resultant matrix.

3.3.6.2.9.17.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R076.

3.3.6.2.9.17.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	Type	Ī	Mode	1	Description	Ī
	Matrix Subtrahend	Matrices Elements		In In		Matrix to be scaled Scale factor	

3.3.6.2.9.17.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Scaled matrix

3.3.6.2.9.17.10.4.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.17.10.4.6 PROCESSING

The following describes the processing performed by this part:

function "-" (Matrix : Matrices;

Subtrahend: Elements) return Matrices is

-- --declaration section-

Answer : Matrices;

-- --begin function "-"

begin

return Answer;

end "-";

3.3.6.2.9.17.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

- 	Name	Туре	Description
Ī	Elements	floating point type	Used to define type of elements in matrix defined by this package
Ì	Col_Indices	discrete type	Used to define second dimension of exported matrix type
	Row_Indices	discrete type	Used to define first dimension of exported matrix type



The following table summarizes the types required by this part and defined in the package specification of Matrix Operations Constrained:

Name	Range	Description
Matrices	N/A 	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.4.8 LIMITATIONS

None.

3.3.6.2.9.17.10.5 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P360-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one in which the diagonal elements equal 1.0 and all other elements equal 0.0. The input matrix must be a square matrix, but the ranges of the individual dimensions do not have to be the same.

3.3.6.2.9.17.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R155.

3.3.6.2.9.17.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	1	Type	1	Mode	1	Description	Ī
Ī							Matrix to be made into an identity matrix	Ī

3.3.6.2.9.17.10.5.4 LOCAL DATA

Data objects:

The following describes the data objects maintained local to this part:

end if;



```
| Name | Type | Description
| Row Indices | Index into first dimension of matrix
3.3.6.2.9.17.10.5.5 PROCESS CONTROL
Not applicable.
3, 3.6.2.9.17.10.5.6 PROCESSING
The following describes the processing performed by this part:
  procedure Set To Identity Matrix (Matrix: out Matrices) is
     --declaration section
     Col : Col Indices;
     Row : Row Indices;
-- --begin procedure Set_To_Identity_Matrix
  begin
     -- make sure input matrix is a square matrix
     if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
        Matrix := (others => (others => 0.0));
        Col := Col Indices'FIRST;
        Row := Row Indices' FIRST;
        Row Loop:
           Ιοου
              --set diagonal element equal to 1
              Matrix(Row, Col) := 1.0;
              exit when Row = Row Indices'LAST;
              Col := Col_Indices'SUCC(Col);
              Row := Row Indices'SUCC(Row);
           end loop Row Loop;
     else
        -- do not have a square matrix
        raise Dimension Frior;
```

end Set To_Identity_Matrix;

3.3.6.2.9.17.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Туре	Description
Elements	floating point type	Used to define type of elements in matrix defined by this package
Col_Indices	discrete type	Used to define second dimension of exported matrix type
Row_Indices	discrete type	Used to define first dimension of exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Matrix_Operations_Constrained:

Name	l	Range	1	Description
Matrices		N/A		Constrained, two-dimensional array of Elements

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

Name	Ī	Description
dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.17.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:



3.3.6.2.9.17.10.6 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P361-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.17.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R156.

3.3.6.2.9.17.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.17.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

•		1	• •	Ī	Mode	•	Description	
1	Matrix	1	Matrices	١	0ut	1	Matrix to be zeroed out	1

3.3.6.2.9.17.10.6.4 LOCAL DATA

None.

3.3.6.2.9.17.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.17.10.6.6 PROCESSING

The following describes the processing performed by this part:

procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

begin

Matrix := (others => (others => 0.0));
end Set_To_Zero_Matrix;

3.3.6.2.9.17.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements defined in this part's ancestral components and used by this part:

Data types:

The following generic types are available to this part and defined at the package specification level of Matrix_Operations_Constrained:

Name	Туре	Description	-
Elements	floating point type	Used to define type of elements in matrix defined by this package	-
Col_Indices	discrete type	Used to define second dimension of exported matrix type	
Row_Indices	discrete type	Used to define first dimension of exported matrix type	

The following table summarizes the types required by this part and defined in the package specification for Matrix Operations Constrained:

Name	Range	Description
Matrices	N/A 	Constrained, two-dimensional array of Elements

3.3.6.2.9.17.10.6.8 LIMITATIONS

None.

3.3.6.2.9.18 DYNAMICALLY SPARSE_MATRIX_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P369-0)

This package defines a dynamically sparse matrix and operations on it. All elements of the matrix are stored, but most of the elements are expected to be 0. Which elements are zero does not have to remain the same. See decomposition section for the operations provided.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.18.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.



3.3.6.2.9.18.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described at the package specification level:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

3.3.6.2.9.18.4 LOCAL DATA

Data types:

The following data types were previously defined at the package specification level:

Name	Range	Description	
Matrices	N/A	Constrained, two-dimensional array of Elements	

3.3.6.2.9.18.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Dynamically_Sparse_Matrix_Operations_Constrained is

end Dynamically Sparse Matrix Operations Constrained;



3.3.6.2.9.18.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.18.8 LIMITATIONS

None.

3.3.6.2.9.18.9 LLCSC DESIGN

None.

3.3.6.2.9.18.10 UNIT DESIGN

3.3.6.2.9.18.10.1 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P370-0)

This procedure sets a square input matrix to an identity matrix. An identity matrix is one where the diagonal elements all equal 1.0, with the remaining elements equaling 0.0.

3.3.6.2.9.18.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	1	Туре		Mode	1	Description	1
1			Matrices				Matrix being made into an identity matrix	1

3.3.6.2.9.18.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:



©

```
| Name | Type | Description
         _____
3.3.6.2.9.18.10.1.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.18.10.1.6 PROCESSING
The following describes the processing performed by this part:
  procedure Set To Identity Matrix (Matrix: out Matrices) is
    --declaration section
    Col : Col_Indices;
    Row : Row Indices;
-- --begin procedure Set_to_Identity_Matrix
  begin
    -- make sure input matrix is a square matrix
    if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
       Matrix := (others => (others => 0.0));
       Col := Col Indices'FIRST;
       Row := Row Indices'FIRST;
       Row Loop:
         Toop
            --set diagonal element equal to 1.0
            Matrix(Row, Col) := 1.0;
```

exit when Row = Row Indices'LAST;
Col := Col Indices'SUCC(Col);
Row := Row_Indices'SUCC(Row);

```
else

raise Dimension_Error;

end if;
```

end loop Row Loop;

end Set to Identity Matrix;

3.3.6.2.9.18.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

]	Name		Range	1	Description	Ī
	Matrices		N/A		Constrained, two-dimensional array of Elements	

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General Vector Matrix Algebra:

Ī	Name	1	Description	Ī
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.18.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:



| Name | When/Why Raised |
| Dimension_Error | Raised if the input matrix is not a square matrix |

3.3.6.2.9.18.10.2 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P371-0)

This procedure sets all elements of an input matrix to zero.

3.3.6.2.9.18.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

| Name | Type | Mode | Description | |
| Matrix | Matrices | In | Matrix to be zeroed out |

3.3.6.2.9.18.10.2.4 LOCAL DATA

None.

3.3.6.2.9.18.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.2.6 PROCESSING

The following describes the processing performed by this part:

procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
begin

Matrix := (others => (others => 0.0));
end Set_to_Zero_Matrix;



3.3.6.2.9.18.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type

The following types are defined in the package specification for Dynamically_-Sparse Matrix Operations Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.18.10.2.8 LIMITATIONS

None.

3.3.6.2.9.18.10.3 ADD TO IDENTITY UNIT DESIGN (CATALOG #P372-0)

This function takes a square input matrix and adds it to an identity matrix by adding 1.0 to all diagonal elements of the input matrix.

3.3.6.2.9.18.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.3.2 LOCAL ENTITIES DESIGN

None.



3.3.6.2.9.18.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type Mode	Description	Ī
Input	Matrices In	Matrix to which is added an identity matrix	

3.3.6.2.9.18.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of adding an identity matrix to the input matrix
Col	Col Indices	N/A	Column index
Row	Row_Indices	N/A	Row index

3.3.6.2.9.18.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.3.6 PROCESSING

The following describes the processing performed by this part:

function Add to Identity (Input: Matrices) return Matrices is

begin

-- -- make sure input is a square matrix if Input'LENGTH(1) = Input'LENGTH(2) then



```
Answer := Input;
      --add "identity" values to diagonal elements
      Col := Col Indices'FIRST;
Row := Row_Indices'FIRST;
      Row Loop:
          Toop
             if Answer(Row, Col) /= 0.0 then
                Answer(Row, Col) := Answer(Row, Col) + 1.0;
                Answer(Row, Col) := 1.0;
             end if:
             exit when Row = Row Indices'LAST;
             Col := Col Indices'SUCC(Col);
             Row := Row Indices' SUCC(Row);
          end loop Row Loop;
   else
      raise Dimension Error;
   end if;
   return Answer;
end Add to Identity;
```

3.3.6.2.9.18.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indi		Used to dimension exported matrix type
Row_Indi	ces discrete type	Used to dimension exported matrix type





The following types are defined in the package specification for Dynamically_-Sparse Matrix Operations Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General Vector Matrix Algebra:

Name		Description
dimension_error	:	Raised by a routine when input received has dimensions incompatible for the type of operation to be performed

3.3.6.2.9.18.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

ī	Name		When/Why Raised	ļ
Ī		•	Raised if the input matrix is not a square matrix	I

3.3.6.2.9.18.10.4 SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P373-0)

This function subtracts a square input matrix from an identity matrix by negating all elements of an input matrix and then adding 1.0 to the elements on the diagonal.

3.3.6.2.9.18.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.4.3 INPUT/OUTPUT





The following table describes this part's formal parameters:

1	Name		Туре		Mode		Description	1
	Input		Matrices		In		Square matrix to be subtracted from an identity matrix	

3.3.6.2.9.18.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices	N/A	Result of subtracting input matrix from an identity matrix
Col	Col Indice		Column index
Row	Row_Indice	s N/A	Row index

3.3.6.2.9.18.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.18.10.4.6 PROCESSING

The following describes the processing performed by this part:

function Subtract from Identity (Input: Matrices) return Matrices is

Col := Col_Indices'FIRST;
Row := Row Indices'FIRST;

Row Loop:



```
loop
            Col Loor:
                for Temp Col in Col Indices loop
                   if Input(Row, Temp Col) /= 0.0 then
                      Answer(Row, Temp Col) := - Input(Row, Temp Col);
                      Answer(Row, Temp Col) := 0.0;
                   end if:
               end loop Col Loop;
            if Answer(Row, Col) /= 0.0 then
               Answer(Row, Col) := Answer(Row, Col) + 1.0;
            else
               Answer(Row, Col) := 1.0;
            end if;
            exit when Row = Row Indices'LAST;
            Col := Col Indices'SUCC(Col);
            Row Row Indices'SUCC(Row);
         end loop Row Loop;
      raise Dimension Error;
   end if;
   return Answer;
end Subtract From Identity;
```

3.3.6.2.9.18.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

else

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices	discrete type	Used to dimension exported matrix type
Row_Indices	discrete type	Used to dimension exported matrix type



The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra:

	Name		Description	1
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.18.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	ŀ	When/Why	Raised	Ī
Dimension_Error	1	Raised if	the input matrix is not a square matrix	1

3.3.6.2.9.18.10.5 "+" UNIT DESIGN (CATALOG #P374-0)

This function adds two sparse m x n matrices, by adding the individual elements of the input matrices taking advantage of the fact that most of the elements of both matrices equal 0.

3.3.6.2.9.18.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:



The following table describes this part's formal parameters:

Na	me	Туре	Mode	Description	
		Matrices Matrices		Sparse matrix to be added Sparse matrix to be added	

3.3.6.2.9.18.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name Type	Value	Description	Ī
Answer Matrices	N/A	Result of adding two input matrices	Ī

3.3.6.2.9.18.10.5.5 PROCESS CONTROL

Not applicable.

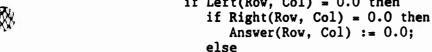
3.3.6.2.9.18.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
function "+" (Left : Matrices;
                Right: Matrices) return Matrices is
     --declaration section
     Answer : Matrices;
-- --begin function "+"
```

begin

```
Row Loop:
   for Row in Row_Indices loop
      Col Loop:
         for Col in Col Indices loop
            if Left(Row, Col) = 0.0 then
```





3.3.6.2.9.18.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations_Constrained:

Ī	Name	Type	Description	
	Elements	floating point type	Data type of elements in exported matrix type	-
Ì	Col_Indices	discrete type	Used to dimension exported matrix type	İ
İ	Row_Indices	discrete type	Used to dimension exported matrix type	Ì

The following types are defined in the package specification for Dynamically_-Sparse_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements



3.3.6.2.9.18.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	1	When/Why	Ra	aised							
Dimension_Error		Raised if	bo	oth matrices	are	not	m	x	n	matrices	

3.3.6.2.9.18.10.6 "-" UNIT DESIGN (CATALOG #P375-0)

This function subtracts two sparse $m \times n$ matrices by subtracting the individual elements of the input matrices, taking advantage of the fact that most of the elements of both matrices equal 0.

3.3.6.2.9.18.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R226.

3.3.6.2.9.18.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.18.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

-	Name	1	Туре		Mode		Description	
			Matrices Matrices				Sparse matrix to be treated as the minuend Sparse matrix to be treated as the subtrahend	

3.3.6.2.9.18.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

						Description	1
Answer	1	Matrices	1	N/A		Result of subtracting two input matrices	Ī



```
3.3.6.2.9.18.10.6.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.18.10.6.6 PROCESSING
The following describes the processing performed by this part:
   function "-" (Left : Matrices;
                 Right : Matrices) return Matrices is
      --declaration section
      Answer : Matrices;
-- --begin function "-"
   begin
      Row Loop:
         for Row in Row Indices loop
            Col Loop:
               for Col in Col Indices loop
                  if Left(Row, Col) = 0.0 then
                     if Right(Row, Col) = 0.0 then
                        Answer(Row, Col) := 0.0;
                     else
                        Answer(Row, Col) := - Right(Row, Col);
                     end if;
                  elsif Right(Row, Col) = 0.0 then
                     Answer(Row, Col) := Left(Row, Col);
                  else
                     Answer(Row, Col) := Left(Row, Col) -
                                             Right(Row, Col);
                  end if;
               end loop Col Loop;
         end loop Row Loop;
     return Answer;
  end "-";
3.3.6.2.9.18.10.6.7 UTILIZATION OF OTHER ELEMENTS
```

UTILIZATION OF ANCESTRAL ELEMENTS:



The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic formal types visible to this part and defined in the package specification for Dynamically_Sparse_Matrix_-Operations Constrained:

Name	Type	Description	Ī
Elements	floating point type	Data type of elements in exported matrix type	1
Col_Indices	discrete type	Used to dimension exported matrix type	İ
Row_Indices	discrete type	Used to dimension exported matrix type	

The following types are defined in the package specification for Dynamically_-Sparse Matrix Operations Constrained:

	Name	1	Range		Description	Ī
	Matrices		Ņ/A		Constrained, two-dimensional array of Elements	1

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_'atrix_Algebra:

ī	Name		Description	-
	dimension_error		Raised by a routine when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.18.10.6.8 LIMITATIONS

None.

3.3.6.2.9.19 SYMMETRIC FULL STORAGE MATRIX_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P398-0)

This package exports operations on a symmetric full storage matrix.

The decomposition for this part is the same as that shown in the Top-Level Design Document.



3.3.6.2.9.19.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R227.

3.3.6.2.9.19.2 LOCAL ENTITIES DESIGN

Subprograms:

There exists a sequence of statements at the end of this package body which are executed when this part is elaborated. The code checks to ensure a square matrix has been instantiated. If not, a Dimension Error exception is raised.

3.3.6.2.9.19.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined at the package specification level:

Data types:

Name	1	Туре	 	Description	Ī
Element	ĺ	floating point type		Data type of elements in exported matrix type	
Col_Ind Row_Ind		discrete type discrete type		Used to dimension exported matrix type Used to dimension exported matrix type	

3.3.6.2.9.19.4 LOCAL DATA

Data types:

The following types are previously defined in this part's package specification:

Ī	Name		Range]	Description	Ī
	Matrices		N/A		Constrained, two-dimensional array of Elements	

3.3.6.2.9.19.5 FROCESS CONTROL

Not applicable.



3.3.6.2.9.19.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Symmetric Full Storage Matrix Operations Constrained is

--processing for Symmetric_Full_Storage_
--Matrix_Operations_Constrained package body

begin

if Matrices'LENGTH(1) /= Matrices'LENGTH(2) then
 raise Dimension_Error;
end if;

end Symmetric_Full_Storage_Matrix_Operations_Constrained;

3.3.6.2.9.19.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following elements are required by this part and defined elsewhere in the TLCSC.

Exceptions:

The following table describes the exceptions required by this part and defined in the package specification for General_Vector_Matrix_Algebra.

1	Name		Description	Ī
	Dimension_Error		Raised by a routine or package when input received has dimensions incompatible for the type of operation to be performed	

3.3.6.2.9.19.8 LIMITATIONS

None.

3.3.6.2.9.19.9 LLCSC DESIGN

None.



3.3.6.2.9.19.10 UNIT DESIGN

3.3.6.2.9.19.10.1 CHANGE ELEMENT UNIT DESIGN (CATALOG #P399-0)

This procedure changes the indicated element of a symmetric matrix, along with its symmetric counterpart.

3.3.6.2.9.19.10.1.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	Туре	Mode	Description
	New_Value Row Col Matrix	Elements Row_Indices Col_Indices Matrices		New value to be placed in the matrix Row in which the value belongs Column in which the value belongs Matrix being updated

3.3.6.2.9.19.10.1.4 LOCAL DATA

None.

3.3.6.2.9.19.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.1.6 PROCESSING

The following describes the processing performed by this part:

-- -- declaration section-

3.3.6.2.9.19.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Constrained:

Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name	Range	Description	1
Matrices	N/A	Constrained, two-dimensional array of Elements	



3.3.6.2.9.19.10.1.8 LIMITATIONS

None.

3.3.6.2.9.19.10.2 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P400-0)

This procedure turns an input matrix into an identity matrix. An identity matrix is one where all elements equal 0.0, except those on the diagonal which equal 1.0. The input matrix must be a square matrix.

3.3.6.2.9.19.10.2.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

}	Name	ŀ	Type	1	Mode	1	Description	1
			Matrices				Matrix to be made into an identity matrix	1

3.3.6.2.9.19.10.2.4 LOCAL DATA

Data objects:

The following table describes the local data maintained by this part:

Ī	Name		Туре		Value		Description	
!	Col Row		Col_Indices Row_Indices		N/A N/A		Index into second dimension of matrix Index into firs: dimension of matrix	

3.3.6.2.9.19.10.2.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.19.10.2.6 PROCESSING The following describes the processing performed by this part: procedure Set To Identity Matrix (Matrix: out Matrices) is --declaration section _____ Col : Col Indices; Row : Row Indices; -- --begin procedure Set to Identity Matrix begin Matrix := (others => (others => 0.0)); Col := Col Indices'FIRST; Row := Row Indices'FIRST; Row Loop: Toop --set diagonal element equal to Matrix(Row, Col) := 1.0; exit when Row = Row Indices'LAST; Col := Col_Indices'SUCC(Col); Row := Row_Indices'SUCC(Row);

3.3.6.2.9.19.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

end Set_To_Identity_Matrix;

end loop Row Loop;

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Constrained:



Name	Type	Description	1
Elements	floating point type	Data type of elements in exported matrix type	
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type	İ

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.2.8 LIMITATIONS

None.

3.3.6.2.9.19.10.3 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P401-0)

This procedure zeros out all elements of an input matrix.

3.3.6.2.9.19.10.3.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

! !		Туре				Descr	pt	ion			1
Ma	atrix	Matrices	 	0ut		Matrix	to	be	zeroed	out	

3.3.6.2.9.19.10.3.4 LOCAL DATA

None.

3.3.6.2.9.19.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Set_To_Zero_Matrix (Matrix : out Matrices) is

begin

Matrix := (others => (others => 0.0));

end Set_To_Zero_Matrix;

3.3.6.2.9.19.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Constrained:

Name	Туре	Description
Elements Col_Indices Row Indices	floating point type discrete type discrete type	Data type of elements in exported matrix type Used to dimension exported matrix type Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

Name Range	Description
Matrices N/A	Constrained, two-dimensional urray of Elements

3.3.6.2.9.19.10.3.8 LIMITATIONS

None.

3.3.6.2.9.19.10.4 ADD_TO_IDENTITY UNIT DESIGN (CATALOG #P402-0)

This function adds an input matrix to an identity matrix, returning the resultant matrix. The addition is performed by adding 1.0 to each diagonal element of the input matrix.

3.3.6.2.9.19.10.4.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1			Туре		Mode	1	Descri	pt:	ion			1
Ī	Input	1	Matrices	1							identity	

3.3.6.2.9.19.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name	I	Туре		Value		Description	
Ī	Answer		Matrices		N/A		Result of adding identity matrix to input matrix	
İ	Col Row	İ	Col_Indices Row_Indices		N/A N/A	İ	Index into second dimension of matrices Index into first dimension of matrices	

3.3.6.2.9.19.10.4.5 PROCESS CONTROL

Not applicable.



3.3.6.2.9.19.10.4.6 PROCESSING The following describes the processing performed by this part: function Add to Identity (Input: Matrices) return Matrices is --declaration section Answer : Matrices; Col : Col_Indices; Row : Row_Indices; -- -- begin function Add to Identity begin Answer := Input; Col := Col Indices'FIRST; Row := Row Indices'FIRST; Access Diagonal Elements: loop Answer(Row,Col) := Answer(Row,Col) + 1.0; exit when Row = Row Indices'LAST; Col := Col_Indices'SUCC(Col); Row := Row Indices'SUCC(Row); end loop Access Diagonal Elements; return Answer;

3.3.6.2.9.19.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

end Add to Identity;

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Constrained:



Name	Type	Description
Elements	floating point type	Data type of elements in exported matrix type
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.4.8 LIMITATIONS

None.

3.3.6.2.9.19.10.5 SUBTRACT FROM IDENTITY UNIT DESIGN (CATALOG #P403-0)

This function subtracts an input matrix from an identity matrix, returning the resultant matrix.

3.3.6.2.9.19.10.5.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description	1
Input	Matrices	In	Matrix to be subtracted from an identity matrix	





3.3.6.2.9.19.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description	Ī
Answer Row S_Col S_Row	Matrices Row_Indices Col_Indices Row_Indices	N/A N/A N/A N/A	Result of adding input matrices Row index into matrix "Symmetric" column index into matrix "Symmetric" row index into matrix; i.e., A(row,col) := A(s_row,s_col)	

3.3.6.2.9.19.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.5.6 PROCESSING

The following describes the processing performed by this part:

function Subtract from Identity (Input : Matrices) return Matrices is

```
-- --declaration section

Answer : Matrices;
Row : Row_Indices;
S_Col : Col_Indices;
S_Row : Row_Indices;
---begin function Subtract_from_Identity
```

begin

-- --handle first diagonal element

```
Answer(Row Indices'FIRST, Col Indices'FIRST) :=
1.0 - Input(Row Indices'FIRST, Col Indices'FIRST);
```

- -- -- will subtract the remaining of the input matrix from an identity matrix -- by doing the following:
- -- o subtracting the nondiagonal elements in the bottom half of the
- -- -- matrix from 0.0,
 - -- o assigning values obtained in the bottom half of the matrix to the
- -- -- symmetric elements in the top half of the matrix, and then
- -- o subtracting the diagonal elements from 1.0
- -- -- S_Col will go across the columns as Row goes down the rows to keep

```
-- track of the column containing the diagonal element
   S Col := Col Indices'SUCC(Col Indices'FIRST);
   Row := Row Indices'SUCC(Row Indices'FIRST);
   Do Every Row Except First:
      loop
         --S Row will go down the rows as Col goes across the columns
         S Row := Row Indices'FIRST;
         Subtract Nondiagonal Elements From Zero:
            for Col in Col Indices' FIRST ...
                       Col Indices' VAL(Row Indices' POS(Row) - 1) loop
               Answer(Row,Col) := - Input(Row,Col);
               Answer(S Row, S Col) := Answer(Row, Col);
               S Row
                         := Row Indices'SUCC(S Row);
            end loop Subtract Nondiagonal Elements From Zero;
         -- subtract diagonal element from 1.0
         Answer(Row, S Col) := 1.0 - Input(Row, S Col);
         exit when Row = Row Indices'LAST;
         S Col := Col Indices'SUCC(S Col);
             := Row Indices'SUCC(Row);
      end loop Do Every Row Except First;
   return Answer;
end Subtract from Identity;
```

3.3.6.2.9.19.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Constrained:

Name	Type	Description	
Elements	floating point type	Data type of elements in exported matrix type	
Col_Indices Row_Indices	discrete type discrete type	Used to dimension exported matrix type Used to dimension exported matrix type	İ



The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name		When/Why	•	
			input matrix is not a square matrix	

3.3.6.2.9.19.10.6 "+" UNIT DESIGN (CATALOG #P404-0)

This function adds two symmetric matrices by adding the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.19.10.6.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	١	Туре	1	Mode		Description
	Left Right		Matrices Matrices		In In		First matrix to be added Second matrix to be added



3.3.6.2.9.19.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Na	me	Туре	Value	Description	
Ans Row	,	Matrices Row_Indices Col_Indices	N/A N/A N/A	Result of adding two input matrices Index into first dimension of matrix Used to keep track of column containing diagonal element for the current row	
S_R 	.ow	Row_Indices	N/A	When used with S_Col, marks the symmetric counterpart to the element being referenced in the bottom half of the array; i.e., A(row,col) = A(s_row,s_col)	

3.3.6.2.9.19.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
function "+" (Left : Matrices;
```

Right: Matrices) return Matrices is

```
-- --declaration section
```

Answer : Matrices;
Row : Row_Indices;
S_Col : Col_Indices;
S_Row : Row_Indices;

```
-- --begin function "+"
```

begin

```
-- --handle first diagonal element
Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
    Left(Row_Indices'FIRST, Col_Indices'FIRST) +
    Right(Row_Indices'FIRST, Col_Indices'FIRST);
```

- -- -- addition calculations will only be carried out on the bottom half
- -- -- of the input matrices followed by assignments to the symmetric
- -- -- elements in the top half of the matrix
- -- -- as Row goes down the rows, S Col will go across the columns to keep

```
--track of the column containing the diagonal element
   S Col := Col Indices'SUCC(Col Indices'FIRST);
   Row := Row Indices'SUCC(Row Indices'FIRST);
   Do All Rows Except First:
      loop
         --as Col goes across the columns, S_Row will go down the rows;
         S Row := Row Indices'FIRST;
         Add Bottom Half Elements:
            For Col in Col Indices'FIRST ..
                       Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop
               --add elements in bottom half of the matrix
               Answer(Row, Col) := Left(Row, Col) + Right(Row, Col);
               --assign value to symmetric element in top half of matrix
               Answer(S Row, S Col) := Answer(Row, Col);
                         := Row_Indices'SUCC(S_Row);
               S Row
            end loop Add Bottom Half Elements;
         -- add diagonal elements together
        Answer(Row, S Col) := Left(Row, S Col) + Right(Row, S Col);
         exit when Row = Row_Indices'LAST;
         S Col := Col Indices'SUCC(S Col);
        Row := Row Indices'SUCC(Row);
     end loop Do All Rows Except First;
   return Answer;
end "+";
```

3.3.6.2.9.19.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations_Constrained:

Ī	Name	Type	 -	Description	-
	Elements	floating point type		Data type of elements in exported matrix type	
İ	Col_Indices Row_Indices	discrete type discrete type		Used to dimension exported matrix type Used to dimension exported matrix type	İ

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_Constrained:

Name	Range	Description
Matrices	N/A 	Constrained, two-dimensional array of Elements

3.3.6.2.9.19.10.6.8 LIMITATIONS

None.

3.3.6.2.9.19.10.7 "-" UNIT DESIGN (CATALOG #P407-0)

This function subtracts two symmetric input matrices by subtracting the individual elements of the input matrices, taking advantage of their symmetricity.

3.3.6.2.9.19.10.7.1 REQUIREMENTS ALLOCATION

See top header.

3.3.6.2.9.19.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.19.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	T	уре		Mode	1	Description	
	Left Right	Ma Ma	trices trices		In In	•	Matrix to be subtracted from Matrix to be used as the subtrahend	•

3.3.6.2.9.19.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer Row S_Col	Matrices Row_Indices Col_Indices	N/A N/A N/A	Result of adding two input matrices Index into first dimension of matrix Used to keep track of column containing diagonal element for the current row
S_Row	Row_Indices 	N/A 	When used with S_Col, marks the symmetric counterpart to the element being referenced in the bottom half of the array; i.e., A(row,col) = A(s_row,s_col)

3.3.6.2.9.19.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.19.10.7.6 PROCESSING

The following describes the processing performed by this part:

function "-" (Left : Matrices;

Right : Matrices) return Matrices is

-- --begin function "-"

begin

```
-- --handle first diagonal element
Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
    Left(Row_Indices'FIRST, Col_Indices'FIRST);
    Right(Row_Indices'FIRST, Col_Indices'FIRST);
```

- -- -- subtraction calculations will only be carried out on the bottom half
- -- -- of the input matrices followed by assignments to the symmetric
- -- -- elements in the top half of the matrix
- -- -- as Row goes down the rows, S_Col will go across the columns to keep
- -- -- track of the column containing the diagonal element
 - S_Col := Col_Indices'SUCC'Cal_Indices'FIRST);
 - Row := Row Indices'SUCC(Row Indices'FIRST);
 - Do All Rows Except First:
 - loop

```
--as Col goes across the columns, S Row will go down the rows;
         S Row := Row Indices'FIRST;
         Subtract_Bottom_Half Elements:
            for Col in Col Indices'FIRST ..
                       Col Indices' VAL(Row Indices' POS(Row) - 1) loop
               --subtract elements in bottom half of the matrix
               Answer(Row, Col) := Left(Row, Col) - Right(Row, Col);
               --assign value to symmetric element in top half of matrix
               Answer(S Row, S Col) := Answer(Row, Col);
                         := Row Indices'SUCC(S Row);
               S Row
            end loop Subtract Bottom Half Elements;
         -- subtract diagonal elements together
         Answer(Row, S Col) := Left(Row, S Col) - Right(Row, S Col);
         exit when Row = Row Indices'LAST;
         S Col := Col Indices'SUCC(S Col);
         Row := Row Indices'SUCC(Row);
      end loop Do All Rows Except First;
   return Answer;
end "-";
```

3.3.6.2.9.19.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one ore more ancestral units:

Data types:

The following table summarizes the generic types visible to this part and defined at the package specification level of Symmetric_Full_Storage_Matrix_-Operations Constrained:

Name	Type	Description
Elemen	ts floating point type	Data type of elements in exported matrix type
Col_Inc		
Row_Inc	dices discrete typ	e Used to dimension exported matrix type

The following table summarizes the types required by this part and defined in the package specification of Symmetric_Full_Storage_Matrix_Operations_-Constrained:

 	Name		Range	 	Description	
	Matrices		N/A		Constrained, two-dimensional array of Elements	

3.3.6.2.9.19.10.7.8 LIMITATIONS

None.

3.3.6.2.9.20 VECTOR_SCALAR_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P422-0)

This package provides a set of functions to multiply and divide each element of a vector by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.20.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the units in this part:

	Requirements
Name	Allocation
"+"	R065 R066

3.3.6.2.9.20.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.20.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types previously in this part's package specification:



_				_
1	Name	Type	Description	
]	Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars	-
İ	Elements2	floating point type	<pre>Type of elements in a vector; Elements2 := Elements1 / Scalars</pre>	İ
j	Scalars	floating point type	Type of value to be used for multiplying and dividing	Ì
İ	Indices	discrete type	Used to dimension vectors	İ
i	Vectors1	array	An array of Elements1	İ
İ	Vectors2	array	An array of Elements2	İ

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description	-
"*"	function	Used to define the operation Elements1 := Elements2 * Scalars	-
"/"	function	Used to define the operation Elements2 := Elements1 / Scalars	İ

3.3.6.2.9.20.4 LOCAL DATA

None.

3.3.6.2.9.20.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.20.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body Vector_Scalar_Operations_Constrained is

end Vector_Scalar_Operations_Constrained;

3.3.6.2.9.20.7 UTILIZATION OF OTHER ELEMENTS

None.



3.3.6.2.9.20.8 LIMITATIONS

None.

3.3.6.2.9.20.9 LLCSC DESIGN

None.

3.3.6.2.9.20.10 UNIT DESIGN

3.3.6.2.9.20.10.1 "*" UNIT DESIGN (CATALOG #P423-0)

This function calculates a scaled vector by multiplying each element of an input vector by a scale factor.

3.3.6.2.9.20.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO65.

3.3.6.2.9.20.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.20.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type Mode	Description
Vector	Vectors2 In	Vector to be scaled
Multiplier	Scalars In	Scale factor

3.3.6.2.9.20.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type		Description	Ī
			Scaled vector	Ī



```
3.3.6.2.9.20.10.1.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.20.10.1.6 PROCESSING
The following describes the processing performed by this part:
   function "*" (Vector : Vectors2;
                Multiplier: Scalars) return Vectors1 is
     --declaration section-
     -----
     Answer : Vectors1;
-- --begin function "*"
  begin
    Process:
       for Index in Indices loop
          Answer(Index) := Vector(Index) * Multiplier;
       end loop Process;
    return Answer;
  end "*";
```

3.3.6.2.9.20.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_Constrained:



_			
-	Name	Туре	Description
- 	Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
	Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
İ	Scalars	floating point type	Type of value to be used for multiplying and dividing
İ	Indices	discrete type	Used to dimension vectors
j	Vectors1	array	An array of Elements1
İ	Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subprograms to Vector_Scalar_Operations_Constrained package:

Name	Туре	Description	
11 * 11	function	Used to define the operation Elements1 := Elements2 * Scalars	

3.3.6.2.9.20.10.1.8 LIMITATIONS

None.

3.3.6.2.9.20.10.2 "/" UNIT DESIGN (CATALOG #P424-0)

This function calculates a scaled vector by dividing each element of an input vector by a scale factor.

3.3.6.2.9.20.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R066.

3.3.6.2.9.20.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.20.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:



1	Name	1	Туре		Mode		Description
	Vector Divisor	1	Vectors1 Scalars		In In		Vector to be scaled Scale factor

3.3.6.2.9.20.10.2.4 LOCAL DATA

Data objects:

The following describes the local data maintained by this part:

Name	1	Type	١	Description	
Answer	1	Vectors2	1	Scaled vector	

3.3.6.2.9.20.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.20.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function "/" (Vector : Vectors1;
```

Divisor : Scalars) return Vectors2 is

```
-- --declaration section-
```

Answer : Vectors2;

```
---begin function Vector_Scalar_Divide
```

begin

```
Process:
```

```
for Index in Indices loop
```

```
Answer(Index) := Vector(Index) / Divisor;
```

end loop Process;

return Answer;

end "/";



3.3.6.2.9.20.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Vector_Scalar_Operations_Constrained:

1	Name	Туре	Description
	Elements1	floating point type	Type of elements in a vector; Elements1 := Elements2 * Scalars
İ	Elements2	floating point type	Type of elements in a vector; Elements2 := Elements1 / Scalars
İ	Scalars	floating point type	Type of value to be used for multiplying and dividing
	Indices	discrete type	Used to dimension vectors
İ	Vectors1	array	An array of Elements1
İ	Vectors2	array	An array of Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subprograms to Vector Scalar Operations Constrained package:

Ī	Name	Type	Description	-
	"/"	function	Used to define the operation Elements2 := Elements1 / Scalars	

3.3.6.2.9.20.10.2.8 LIMITATIONS

None.

3.3.6.2.9.21 MATRIX_SCALAR_OPERATIONS_CONSTRAINED PACKAGE DESIGN (CATALOG #P428-0)

This package provides a set of functions which will scale a matrix by multiplying or dividing each element of the matrix by a scale factor.

The decomposition for this part is the same as that shown in the Top-Level Design Document.



3.3.6.2.9.21.1 REQUIREMENTS ALLOCATION

The following table describes the allocation of requirements to the parts in this LLCSC:

 N	Jame	Requirements Allocation	
"*	PAS .	R073 R074	

3.3.6.2.9.21.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.21.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described in this part's package specification:

Data types:

Name	Туре	Type Description		
Elements1	floating point type	Type of elements in an array		
Elements2	floating point type	Type of elements in an array		
Scalars	floating point type	Data type of objects to be used as rultipliers and divisors		
Col Indices	discrete type	Used to dimension second dimension of matrices		
Row Indices	discrete type	Used to dimension first dimension of matrices		
Matrices1	array	Two dimensional matrix with elements of type Elements1		
Matrices2	array	Two dimensional matrix with elements of type Elements2		

Subprograms:

Name	Type	Description
пжп	function	Function to define the operation
"/"	function	Function to define the operation Elements2 / Scalars := Elements1

V.V

3.3.6.2.9.21.4 LOCAL DATA

None.

3.3.6.2.9.21.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.21.6 PROCESSING

The following describes the processing performed by this part:

separate (General Vector Matrix Algebra)
package body Matrix Scalar Operations Constrained is

end Matrix_Scalar_Operations_Constrained;

3.3.6.2.9.21.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

3.3.6.2.9.21.8 LIMITATIONS

None.

•

3.3.6.2.9.21.9 LLCSC DESIGN

None.

3.3.6.2.9.21.10 UNIT DESIGN

3.3.6.2.9.21.10.1 "*" UNIT DESIGN (CATALOG #P429-0)

This function calculates a scaled matrix by multiplying each element of an input matrix by a scalar.

3.3.6.2.9.21.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R073.

3.3.6.2.9.21.10.1.2 LOCAL ENTITIES DESIGN

None.



3.3.6.2.9.21.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode	Description
Matrix	Matrices1	In	Matrix to be scaled
Multiplier	Scalars	In	Scale factor

3.3.6.2.9.21.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name Type	Description	Ī
Answer Matrices2		l

3.3.6.2.9.21.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.21.10.1.6 PROCESSING

The following describes the processing performed by this part:

function "*" (Matrix : Matrices1;

Multiplier : Scalars) return Matrices2 is

```
-- --declaration section-
```

Answer : Matrices2;

```
-- --begin function "*"
```

begin

```
Row Loop:
For Row in Row_Indices loop
```

Col_Loop:
 for Col in Col Indices loop



Answer(Row, Col) := Matrix(Row, Col) * Multiplier;

end loop Col_Loop;

end loop Row Loop;

return Answer:

end "*";

3.3.6.2.9.21.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF ANCESTRAL ELEMENTS:

The following tables describe the elements used by this part but defined in one or more ancestral units:

Data types:

The following table summarizes the generic types required by this part and defined at the package specification level of Matrix Scalar Operations:

Name	Туре	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col Indices	discrete type	Used to dimension second dimension of matrices
Row Indices	discrete type	Used to dimension first dimension of matrices
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix_Scalar_Operations_Constrained package.

Name	Type	Description	
11*11	function	Function to define the operation Elements1 * Scalars := Elements2	



3.3.6.2.9.21.10.1.8 LIMITATIONS

None.

3.3.6.2.9.21.10.2 "/" UNIT DESIGN (CATALOG #P430-0)

This function calculates a scaled matrix by dividing each element of an input matrix by a scale factor.

3.3.6.2.9.21.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R074.

3.3.6.2.9.21.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.21.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode	Description
Matrix	Matrices2	In	Matrix to be scaled
Divisor	Scalars	In	Scale factor

3.3.6.2.9.21.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name Type	Description	
Answer Matrices1		

3.3.6.2.9.21.10.2.5 PROCESS CONTROL

Not applicable.

```
3.3.6.2.9.21.10.2.6 PROCESSING
The following describes the processing performed by this part:
   function "/" (Matrix : Matrices2;
                 Divisor : Scalars) return Matrices1 is
      --declaration section-
      Answer : Matrices1;
-- --begin function "/"
   begin
      Row Loop:
         for Row in Row Indices loop
            Col_Loop:
                for Col in Col Indices loop
                  Answer(Row, Col) := Matrix(Row, Col) / Divisor;
               end loop Col_Loop; .
         end loop Row Loop;
      return Answer;
   end "/";
3.3.6.2.9.21.10.2.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF ANCESTRAL ELEMENTS:
The following tables describe the elements used by this part but defined in one
or more ancestral units:
Data types:
```

The following table summarizes the generic types required by this part and defined at the package specification level of Matrix Scalar Operations:

Name	Туре	Description
Elements1	floating point type	Type of elements in an array
Elements2	floating point type	Type of elements in an array
Scalars	floating point type	Data type of objects to be used as multipliers and divisors
Col Indices	discrete type	Used to dimension second dimension of matrices
Row Indices	discrete type	Used to dimension first dimension of matrices
Matrices1	array	Two dimensional matrix with elements of type Elements1
Matrices2	array	Two dimensional matrix with elements of type Elements2

Subprograms and task entries:

The following table describes the subprograms required by this part and defined as generic formal subroutines to the Matrix_Scalar_Operations_Constrained package.

Name	Type	Description	1
"*"	function	Function to define the operation Elements1 * Scalars := Elements2	
"/"	function	Function to define the operation Elements2 / Scalars := Elements1	İ

3.3.6.2.9.21.10.2.8 LIMITATIONS

None.

3.3.6.2.9.22 VECTOR_MATRIX_MULTIPLY_UNRESTRICTED PACKAGE DESIGN (CATALOG #P437-0)

This package contains a function which multiplies a $1 \times m$ vector by an $m \times n$ matrix producing a $1 \times n$ vector. If the length of the vector is not the same as the length of the first dimension of the matrix a DIMENSION_ERROR exception is raised. None of the ranges need to be the same.

The function in this package can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

The following table lists the catalog numbers for subunits contained in this part:



ī	Name	1	Catalog _	#
Ţ	***		P1051-0	

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.22.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.9.22.2 LOCAL ENTITIES DESIGN

Subprograms:

This package contains code which checks the lengths of the indices used to instantiate the package to ensure the sizes of the input vector, input matrix, and output vector are compatible with the operation to be performed.

3.3.6.2.9.22.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of General_Vector_Matrix_Algebra:

Data types:

The following table describes the generic formal types required by this part:

Name	Туре	Description
Input_Vector_Elements	floating point type	Type of elements in the input vector
Matrix_Elements	floating point type	Type of elements in the input matrix
Output_Vector_Elements	floating point type	Type of elements in the output vector
Col_Indices	discrete type	Used to dimension second dimension of input matrix
Row_Indices	discrete type	Used to dimension first dimension of input matrix
Input Vector Indices	discrete	Used to dimension input vector
Output Vector_Indices	discrete	Used to dimension output vector
Input_Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector



Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description	
"*" 	function	Function * ing the operation Input_V x_Elements * Matrix_Elements := Output_Vector_Elements	
"+" 	function	Function defining the operation Output_Vector_Elements + Output_Vector_Elements := Output_Vector_Elements	

FORMAL PARAMETERS:

The following table describes the formal parameters for the "*" unit contained in this part:

Name	Туре	Description
Vector Matrix	<pre>Input_Vectors Input_Matrices</pre>	1xm vector to be used in the calculation mxn matrix to be used in the calculation

3.3.6.2.9.22.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by the unit in this part:

Name	Type	Value	Description
Answer M_V	Output Vectors Input_Vector_ Indices	N/A N/A	Result vector being calculated Index into the 1 x m input vector
N_A	Output_Vector_ Indices	N/A	Index into the n x 1 output vector
N I	Col_Indices	N/A	Column index into the m x n input matrix
M	Row_Indices	N/A	Row index into the m x n input matrix

```
3.3.6.2.9.22.5 PROCESS CONTROL
Not applicable.
3.3.6.2.9.22.6 PROCESSING
The following describes the processing performed by this part:
separate (General Vector Matrix Algebra)
package body Vector Matrix Multiply Unrestricted is
   function "*" (Vector : Input Vectors;
                Matrix : Input Matrices) return Output Vectors is
        ______
     --declaration section-
     _____
     Answer : Output Vectors := (others => 0.0);
     M V : Input Vector Indices;
           : Output Vector Indices;
     NA
     N
           : Col_Indices;
           : Row Indices;
-- --begin function "*"
  begin
     N A := Output Vector Indices'FIRST;
     N := Col Indices'FIRST;
     N Loop:
        loop
           M V := Input Vector Indices'FIRST;
           M := Row Indices'FIRST;
           M Loop:
              loop
                 Answer (N A) := Answer(N A) + Vector(M V) * Matrix(M, N);
                 exit when M = Row Indices'LAST;
                 M := Row Indices'SUCC(M);
                 M V := Input Vector Indices'SUCC(M V);
              end loop M_Loop;
           exit when N = Col Indices'LAST;
           N := Col Indices'SUCC(N);
           N A := Output Vector Indices'SUCC(N A);
        end loop N Loop;
     return Answer:
```

begin

- -- -- make sure package has been instantiated with the correct dimensions;
 -- -- the following dimensions are expected: [1xm] * [mxn] => [1xn]
 - if Input_Vectors'LENGTH /= Input_Matrices'LENGTH(1) or --m's not equal Input_Matrices'LENGTH(2) /= Output_Vectors'LENGTH then --n's not equal

raise Dimension_Error;

end if;

end Vector_Matrix_Multiply_Unrestricted;

3.3.6.2.9.22.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of General_Vector_Matrix_Algebra:

1	Name	<u> </u>	Description	Ī
	Dimension_Error		Raised by a routine or package when input received has dimensions incompatible with the type of operation to be performed	

3.3.6.2.9.22.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ī	Name		When/Why Raised	Ī
	Dimension_Error		Raised if the sizes of the data objects are incompatible for the multiplication operation	



3.3.6.2.9.22.9 LLCSC DESIGN

None.

3.3.6.2.9.22.10 UNIT DESIGN

None.

3.3.6.2.9.23 ABA TRANS_DYNAM_SPARSE_MATRIX_SQ_MATRIX PACKAGE DESIGN (CATALOG #P1066-0)

This package contains a function which does an ABA transpose multiply on a dynamically sparse matrix $(m \times n)$ and a square $(n \times n)$ matrix, yielding a square matrix. The first multiply (A*B) is constrained and the second (AB*transpose A) is restricted.

The following table lists the catalog numbers for subunits contained in this part:

Name	I	Catalog _#	1
ABA_Transpose	1	P1067-0	1



The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.23.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.23.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.23.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:



Name	Type	Description
A_Elements	floating point type	Type of element in the dymamically sparse input matrix
B_Elements	floating point type	Type of element in the square input matrix
C_Elements	floating point type	Type of element in the output vector
M_Indices	discrete type	Used to dimension the 1st dimension of the sparse input matrix
N_Indices 	discrete type 	Used to dimension the 2nd dimension of the sparse matrix and both dimensions of the square matrix
A_Matrices	array	Data type of the dynamically sparse input matrix
B_Matrices	array 	Data type of the square input matrix
C_Matrices	array	Data type of the output matrix

The following table summarizes the generic formal subroutines required by this part:

Ī	Name		Туре	1	Description	
	"*"	-	function		Function defining the operation A_Elements * B_Elements := C Elements	-
	п×п		function		Function defining the operation C_Elements * A_Elements := C_Elements	

3.3.6.2.9.23.4 LOCAL DATA

None.

3.3.6.2.9.23.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.23.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra)
package body ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix is

Answer : A_Elements;







```
begin
   If Left = 0.0 then
      Answer := 0.0;
      Answer := Left * A Elements( Right );
   end if:
   return Answer;
end Sparse Left Multiply;
function Sparse Right Multiply( Left : A Elements;
                                  Right: A Elements ) return C Elements is
   Answer : C Elements;
begin
   If Right = 0.0 then
      Answer := 0.0;
      Answer := C Elements( Left * Right );
   end if;
   return Answer;
end Sparse Right Multiply;
function Matrix Multiply is new Matrix Matrix Multiply Restricted
   ( Left Elements => A Elements,
     Right Elements -> B Elements,
     Output Elements => A Elements,
     M Indices => M Indices,
     N Indices => N Indices,
P Indices => N Indices,
Left Matrices => A Matrices,
Right Matrices => B Matrices,
     Output Matrices => A_Matrices,
                      => Sparse Left Multiply );
function Matrix Transpose Multiply is new
   Matrix Matrix Transpose Multiply Restricted
      ( Left Elements => A Elements,
        Right Elements => A Elements,
        Output Elements => C Elements,
        M_Indices => M_Indices,
                         => N Indices,
        N Indices
        P<sup>-</sup>Indices
                        => M_Indices,
        Left Matrices => A Matrices,
        Right Matrices => A Matrices,
        Output Matrices => C Matrices,
                          => Sparse Right_Multiply );
function ABA Transpose( A : A Matrices;
                         B : B Matrices )
                               return C_Matrices is
```

3.3.6.2.9.23.7 UTILIZATION OF OTHER ELEMENTS

end ABA Trans Dynam Sparse Matrix Sq Matrix;

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in this top-level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in ancestral units:

Name	Type	Source Description
Matrix Matrix Multiply Restricted Matrix Matrix Transpose Multiply	generic function generic function	GVMA Used to multiply the sparse matrix by the square matrix GVMA Used to multiply the product of the first operation by the transpose of the sparse matrix

3.3.6.2.9.23.8 LIMITATIONS

None.



3.3.6.2.9.23.9 LLCSC DESIGN

None.

3.3.6.2.9.23.10 UNIT DESIGN

None.

3.3.6.2.9.24 ABA TRANS VECTOR SQ MATRIX PACKAGE DESIGN (CATALOG #P1068-0)

This package contains a function which does an ABA transpose multiply on a vector $(1 \times m)$ and a square $(m \times m)$ matrix, yielding a scalar value.

The following table lists the catalog numbers for subunits contained in this part:

Name		Catalog _	#	-
ABA_Transpose	1	P1069-0		-

The decomposition for this part is the same as that shown in the Top-Level Design Document.



3.3.6.2.9.24.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.24.2 LOCAL ENTITIES DESIGN

Subprograms:

The following table summarizes the sabroutines which are local to this part:

Name	Туре	Description
Multiply_VM	function	Function defining the operation Vector_ Elements * Matrix_Elements := Vector_ Elements
Multiply_VV	function	Function defining the operation Vector
Vector_Matrix_ Multiply 	function	Function defining a vector matrix multipli- cation Input_Vectors * Input_Matrices := Output_Vectors. Instantiation of GVMA. Vector Matrix Multiply Restricted
Vector_Vector_ Multiply 	function	Function defining a vector vector multiply (dot product) Vectors * Vectors := Scalars Instantiation of GVMA.Dot_Product_ Operations_Restricted



3.3.6.2.9.24.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Vector_ Elements	floating point type	Type of element in the input vector.
Matrix_ Elements	floating point type	Type of element in the sqaure input matrix.
Scalars	floating point type	Type of element in the output scalar
Indices	discrete type	Used to dimension the input vector and both dimensions of the input matrix
Vectors	array	Data type of the input vector
Matrices	array	Data type of the square input matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

1	Name	1	Туре		Description	
Ī	11 🛠 11		function		Function defining the operation Vector_Elements * Matrix_Elements := Vector_Elements	
	11 * 11		function		Function defining the operation Vector_Elements * Vector_Elements := Scalars	

3.3.6.2.9.24.4 LOCAL DATA

None.

3.3.6.2.9.24.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.24.6 PROCESSING

The following describes the processing performed by this part:

separate (General Vector Matrix Algebra)
package body ABA_Trans_Vector_Sq_Matrix is

function Multiply_VM(Left : Vector_Elements;





```
Right: Matrix Elements) return Vector Elements is
begin
   return Left * Vector Elements( Right );
end Multiply VM;
function Multiply VV( Left : Vector Elements;
                      Right: Vector Elements ) return Scalars is
   return Scalars( Left ) * Scalars( Right );
end Multiply VV;
function Vector Matrix Multiply is new Vector Matrix Multiply Restricted
         ( Input Vector Elements => Vector Elements,
           Matrix Elements => Matrix Elements.
           Output Vector Elements => Vector Elements,
                                 => Indices.
           Indices1
                                 => Indices.
           Indices2
                               -> Vectors,
-> Matrices,
-> Vectors,
           Input Vectors
           Input_Matrices
Output_Vectors
           11 + 11
                                  => Multiply VM );
function Vector Vector Multiply is new Dot_Product_Operations_Restricted
            ( Left Elements => Vector Elements,
              Right Elements => Vector Elements.
              Result Elements => Scalars,
              Indices => Indices,
Left Vectors => Vectors,
              Right Vectors => Vectors,
                            => Multiply VV );
function ABA Transpose( A : Vectors;
                        B: Matrices) return Scalars is
   Partial Answer: Vectors;
   Answer
           : Scalars;
begin
  - multiply A * B -
  Partial Answer := Vector Matrix Multiply( Vector => A,
                                              Matrix => B );
   - multiply AB * transpose of A -
  Answer := Vector_Vector_Multiply( Left => Partial_Answer,
                                                             413
```

Right \Rightarrow A);

return Answer;

end ABA Transpose;

end ABA_Trans_Vector_Sq_Matrix;

3.3.6.2.9.24.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in this top-level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in ancestral units:

Name	Type	Source Description
Vector Matrix Multiply_ Restricted Dot Product Multiply_ Restriced	generic function generic function	GVMA Used to multiply the input vector by the square matrix GVMA Used to multiply the product of the first operation by the transpose of the input vector

3.3.6.2.9.24.8 LIMITATIONS

None.

3.3.6.2.9.24.9 LLCSC DESIGN

None.

3.3.6.2.9.24.10 UNIT DESIG"

None.

3.3.6.2.9.25 ABA TRANS VECTOR SCALAR PACKAGE DESIGN (CATALOG #P1070-0)

This package contains a function which does an ABA transpose multiply on a vector $(m \times 1)$ and a scalar value, yielding a square $(m \times m)$ matrix.

The following table lists the catalog numbers for subunits contained in this part:



Name	l	Catalog _#	
ABA_Transpose		P1071-0	Ī

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.25.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.25.2 LOCAL ENTITIES DESIGN

Packages:

The following table summarizes the packages which are local to this part:

Name	Type	Description	1
VS_Opns	package	Package defining Vector Scalar operations. Only multiply operator is used. Instantiation of GVMA.Vector_Scalar_Operations_ Constrained	

Subprograms:

The following table summarizes the subroutines which are local to this part:

Name	Type	Description
Multiply	VS function	Function defining the operation Vector Elements * Scalars := Vector Elements. Used in instantiation of Vector Scalar Operations Constrained
Divide_V	S function	
VV_Trans Multipl		



3.3.6.2.9.25.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Туре	Description
Vector_ Elements	floating point type	Type of element in the input vector.
Matrix_ Elements	floating point type	Type of element in the square output matrix.
Scalars	floating point type	Type of element in the output scalar
Indices	discrete type	Used to dimension the input vector and the square output matrix
Vectors	array	Data type of the input vector
Matrices	array	Data type of the square output matrix

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Ī	Name	١	Туре		Description	•
	11 * 11	1	function		Function defining the operation Vector_Elements * Scalars := Vector Elements	
	"*"	İ	function		Function defining the operation Vector Elements * Vector Elements := Matrix Elements	

3.3.6.2.9.25.4 LOCAL DATA

None.

3.3.6.2.9.25.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.25.6 PROCESSING

The following describes the processing performed by this part:

separate (General Vector Matrix Algebra) package body ABA Trans Vector Scalar is

-- -- Operators provided for instantiations -

```
CAMP Software Detailed Design Document
  ------
   function Multiply_VS( Left : Vector_Elements;
                         Right: Scalars) return Vector Elements is
   begin
      return Left * Vector Elements( Right );
   end Multiply VS;
-- -- This operator is not used, but is required for the instantiation. -
-- -- It is "dummied" out to make it as small as possible.
   function Divide_VS( Left : Vector Elements;
                      Right: Scalars) return Vector Elements is
   begin
     return Left;
   end Divide VS;
   function Multiply_VV( Left : Vector_Elements;
                        Right : Vector Elements ) return Matrix Elements is
   begin
      return Matrix_Elements( Left ) * Matrix_Elements( Right );
   end Multiply VV;
-- -- Instantiations for ABA transpose -
   package VS Opns is new Vector Scalar Operations Constrained
               ( Elements1 => Vector Elements,
                Element 2 => Vector Elements,
                 Scalars => Scalars,
                Indices => Indices,
Vectors1 => Vectors,
                Vectors2 => Vectors
                "*" => Multipl_VS,
"/" => Divide_VS);
  use VS Opns;
  function VV Transpose Multiply is new
     Vector Vector Transpose Multiply Restricted
         ( Left Vector Elements => Vector Elements,
          Right_Vector_Elements => Vector_Elements,
          Matrix_Elements => Matrix_Elements,
Indices1 => Indices,
          Indices1
Indices2
                               => Indices,
          Indices2
Left_Vectors
Right_Vectors
                               => Vectors,
                             => Vectors,
         Matrices
                               => Matrices,
```

=> Multiply_VV);



11 🛧 11

3.3.6.2.9.25.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in this top-level component:

Packages:

The following table summarizes the external packages required by this part:

	Name	Type	Source	Description	1
	Vector_Scalar_ Operations_ Constrained	generic package	GVMA	Package allowing operation Vectors * Scalars := Vectors.	

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

-	Name		Туре	Source	Description	
	Matrix_Matrix_ Transpose_ Multiply		generic function	GVMA	Function allowing the operation vector * transpose vector := Matrices.	



3.3.6.2.9.25.8 LIMITATIONS

None.

3.3.6.2.9.25.9 LLCSC DESIGN

None.

3.3.6.2.9.25.10 UNIT DESIGN

None.

3.3.6.2.9.26 COLUMN MATRIX OPERATIONS PACKAGE DESIGN (CATALOG #P1072-0)

This package defines a column matrix which contains a column vector which is set on one of the columns of the matrix and a diagonal, which can only have the values of 1 or 0 on the diagonal. It provides operations on that type. See the top level decomposition section for a list of the operations provided.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.2.9.26.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.26.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.26.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description	1
Vector_ Elements Indices Vectors	floating point type discrete type array	Type of element in the column matrix's column vector Used to dimension the column matrix and the vector in the column matrix Data type of the vector in the column matrix	



3.3.6.2.9.26.4 LOCAL DATA

None.

3.3.6.2.9.26.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.26.6 PROCESSING

The following describes the processing performed by this part:

separate (General_Vector_Matrix_Algebra) package body Column_Matrix_Operations is

end Column Matrix Operations;

3.3.6.2.9.26.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.26.8 LIMITATIONS

None.

3.3.6.2.9.26.9 LLCSC DESIGN

None.

3.3.6.2.9.26.10 UNIT DESIGN

3.3.6.2.9.26.10.1 SET_DIAGONAL_AND_SUBTRACT_FROM_IDENTITY UNIT DESIGN (CATALOG #P1073-0)

This function provides the initialization of a column matrix with values of a vector (to be subtracted from the identity matrix) and the diagonal to be set to 1's.

3.3.6.2.9.26.10.1.1 REQUIREMENTS ALLOCATION

This part meets requirement R.

3.3.6.2.9.26.10.1.2 LOCAL ENTITIES DESIGN

None.



3.3.6.2.9.26.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	Mode Type Desc	ription	
	Column Active_Column	in Vectors The v in Indices Value	ector to be set on the designating which co	he specified column olumn is to be set

3.3.6.2.9.26.10.1.4 LOCAL DATA

Data objects:

The following table describes the objects maintained by this part:

1		•	Туре	 	Description	Ī
•	Answer	1	Column	_Matrices	The resultant column matrix	1

(A)

3.3.6.2.9.26.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.26.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Set_Diagonal and Subtract_from_Identity
  ( Column : Vectors;
    Active_Column : Indices ) return Column_Matrices is
Answer : Column Matrices;
```

begin



3.3.6.2.9.26.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.26.10.1.8 LIMITATIONS

None.

3.3.6.2.9.26.10.2 ABA TRANSPOSE UNIT DESIGN (CATALOG #P1075-0)

This package contains a function which does an ABA transpose multiply on a column matrix (m \times m) and a square (m \times m) matrix, yielding a square matrix (m \times m). The matrix multiplies (A*B) and (AB * transpose A) are restricted.

3.3.6.2.9.26.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R.

3.3.6.2.9.26.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.26.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Ī	Name	Туре	Description	Ī
	B_Matrix_ Elements C_Matrix_ Elements B_Matrices C_Matrices	floating point type floating point type array array	Type of element in the square input matrix Type of element in the output vector Data type of the square input matrix Data type of the output matrix	

Subprograms:

The following table describes the generic formal subroutines required by this part:



Ī	Name		Type		Description	Ī
	ü¥ü		function		Function defining the operation Vector_Elements * B_Matrix_Elements := B_Matrix_Elements	

FORMAL PARAMETERS:

The following table describes the formal parameters for this part:

Nап	ne Type	Description
A	Column_Matrices	The input column matrix
В	B_Matrices	The input square matrix

3.3.6.2.9.26.10.2.4 LOCAL DATA

None.

3.3.6.2.9.26.10.2.5 PROCESS CONTROL

Not applicable.

else

3.3.6.2.9.26.10.2.6 PROCESSING

The following describes the processing performed by this part:

function ABA Transpose(A : Column Matrices;

```
B: B Matrices ) return C Matrices is
            : C Matrices;
Answer
Temp_Vector : Vectors := A.Col Vector;
if A.Diagonal then
   Temp Vector( A.Active Column ) := Temp Vector( A.Active Column ) - 1.0;
   M Loop:
     For Row in Indices loop
         P Loop:
            For Col in Indices loop
               Answer( Row, Col ) := C Matrix Elements(
                  Temp Vector( Row ) * Temp Vector( Col ) *
                  B( A.Active_Column, A.Active_Column )
                  Temp_Vector( Col ) * P( A.Active_Column, Row ) +
                  Temp Vector( Row ) * B( A.Active Column, Col ) +
                  B( Row, Col ) );
            end loop P Loop;
      end loop M Loop;
```

3.3.6.2.9.26.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.9.26.10.2.8 LIMITATIONS

None.

3.3.6.2.9.26.10.3 ABA SYMM TRANSPOSE UNIT DESIGN (CATALOG #P1076-0)

This package contains a function which does an ABA transpose multiply on a column matrix $(m \times m)$ and a symmetric $(m \times m)$ matrix, yielding a symmetric matrix $(m \times m)$. The matrix multiplies (A*B) and (AB * transpose A) are restricted.

3.3.6.2.9.26.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R.

3.3.6.2.9.26.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.9.26.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:



B Matrix	Ī	Name	Туре	Description	Ī
C Matrices array Data type of the output matrix		Elements C_Matrix_ Elements B_Matrices	point type floating point type array	Type of element in the output vector Data type of the square input matrix	

The following table describes the generic formal subroutines required by this part:

Na	me		Type		Description	1
"*	**		function		Function defining the operation Vector_Elements * B_Matrix_Elements := B_Matrix_Elements	-

FORMAL PARAMETERS:

]	Name	Туре	Description
ļ	A	Column_Matrices	The input column matrix
	В	B_Matrices	The input symmetric matrix

The following table describes the formal parameters for this part:

3.3.6.2.9.26.10.3.4 LOCAL DATA

None.

3.3.6.2.9.26.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.2.9.26.10.3.6 PROCESSING

The following describes the processing performed by this part:

function ABA Symm Transpose(A : Column Matrices;

B : B Matrices) return C Matrices is

Answer : C Matrices;



None.

```
Last : Indices;
      Temp Vector : Vectors := A.Col Vector;
   begin
      Last := Indices'LAST;
      if A.Diagonal then -- Diagonal value is 1 --
         Temp Vector( A.Active Column ) := Temp Vector( A.Active Column ) - 1.0;
        M Loop:
           For Row in Indices loop
              P Loop:
                 - Calculate values -
                 For Col in Row .. Indices'LAST loop
                    Answer( Row, Col ) := C_Matrix_Elements(
   Temp_Vector( Row ) * Temp_Vector( Col ) *
                       B( A.Active_Column, A.Active Column )
                       Temp Vector(Col) * B( A.Active Column, Row) +
                       Temp_Vector( Row ) * B( A.Active_Column, Col ) +
                       B( Row, Col ) );
                    - Assign calculated value to corresponding -
                    - lower triangular position
                    -----
                    Answer( Col, Row ) := Answer( Row, Col );
                 end loop P Loop;
           end loop M Loop;
      else
                              -- diagonal value is 0 --
        M1 Loop:
           For Row in Indices loop
              P1 Loop:
                 - Calculate values -
                 For Col in Row .. Indices'LAST loop
                    Answer( Row, Col ) := C_Matrix_Elements(
                       A.Col_Vector( Row ) * A.Col_Vector( Col ) * B( A.Active_Column, A.Active_Column ) );
                    - Assign calculated value to corresponding -
                    - lower triangular position
                    ------
                    Answer( Col, Row ) := Answer( Row, Col );
                 end loop P1 Loop;
           end loop M1_Loop;
     end if;
                               -- Diagonal value is 0 --
     return Answer;
  end ABA Symm Transpose;
3.3.6.2.9.26.10.3.7 UTILIZATION OF OTHER ELEMENTS
```

3.3.6.2.9.26.10.3.8 LIMITATIONS

None.

3.3.6.2.10 UNIT DESIGN

3.3.6.2.10.1 MATRIX MATRIX MULTIPLY RESTRICTED UNIT DESIGN (CATALOG #P441-0)

This function multiplies an $m \times n$ matrix by an $n \times p$ matrix, returning an $m \times p$ matrix.

The result of this operation is defined as follows:

$$a(i,j) := b(i,k) * c(k,j)$$

3.3.6.2.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R077.

3.3.6.2.10.1.2 LOCAL ENTITIES DESIGN

None.



3.3.6.2.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification for General Vector Matrix Algebra.

Data types:

The following table describes the generic formal types required by this part:



Name	Type	Description
Left_Elements	floating point type	Data type of elements in left input matrix
Right_Elements	floating point type	Data type of elements in right input matrix
Output_Elements	floating point type	Data type of elements in output matrix
M_Indices	discrete type	Used to dimension first dimension of left input matrix and output matrix
N_Indices	discrete type	Used to dimension second dimension of left input matrix and first dimension of right input matrix
P_Indices	discrete type	Used to dimension second dimension of right input matrix and output matrix
Left Matrices	array	Data type of left input matrix
Right Matrices	array	Data type of right input matrix
Output Matrices	arraý	Data type of output matrix

The following table describes the generic formal subroutines required by this part. To tailor this function to handle sparse matrices, the formal subroutines should be set up to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description
n*n	function	Function defining the operation Left Elements * Right Elements := Output Elements
"+"	function	Function defining the operation Output_Elements + Output_Elements := Output_Elements

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Matrices	In	m x n matrix
Right	Right_Matrices	In	n x p matrix

3.3.6.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:





```
Name | Type | Value | Description
| Answer | Output Matrices | N/A | Result matrix
3.3.6.2.10.1.5 PROCESS CONTROL
Not applicable.
3.3.6.2.10.1.6 PROCESSING
The following describes the processing performed by this part:
separate (General Vector Matrix Algebra)
function Matrix Matrix Multiply Restricted (Left : Left Matrices;
               Right: Right Matrices) return Output Matrices is
-- --declaration section-
   Answer : Output Matrices;
--begin of function Matrix_Matrix_Multiply_Restricted
begin
  Answer := (others \Rightarrow (others \Rightarrow 0.0));
  M Loop:
     for M in M Indices loop
        P Loop:
           for P in P Indices loop
              N Loop:
                 for N in N Indices loop
                    Answer(M, P) := Answer(M, P) +
                                    Left(M, N) * Right(N, P);
                 end loop N Loop;
           end loop P Loop;
     end loop M Loop;
  return Answer;
end Matrix_Matrix_Multiply_Restricted;
```

3.3.6.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.1.8 LIMITATIONS

None.

3.3.6.2.10.2 MATRIX VECTOR MULTIPLY RESTRICTED UNIT DESIGN (CATALOG #P436-0)

This function multiplies an m \times n matrix by an n \times 1 vector producing an m \times 1 vector.

The result of this operation is defined as follows:

$$a(i) := b(i,j) * c(j)$$

3.3.6.2.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R069.

3.3.6.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following table describes this part's generic parameters which were previously described in the package specification of General_Vector_Matrix_-Algebra:

Data types:

The following table describes the generic formal types required by this part:

1	
10	<u>u</u>
1	
100	

Name	Type	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Indices1	discrete type 	Used to dimension first dimension of input matrix and to dimension the output vector
Indices2	discrete type 	Used to dimension second dimension of input matrix and to dimension the input vector
Input Matrices	array	Data type of input matrix
Input_Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Type	Description	-
11*11	function	Function defining the operation	Ī
		Matrix_Elements * Input_Vector_Elements := Output Vector Elements	ļ
"+"	function	Function defining the operation	
		Output_Vector_Elements +	
		Output_Vector_Elements :=	1
		Output_Vector_Elements	1

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	I	Туре	I	Mode	1	Description	
Matrix Vector		Input_Matrices Input_Vectors		In In		Matrix to be used as the multiplicand Vector to be used as the multiplier	

3.3.6.2.10.2.4 LOCAL DATA

Data objects:



end loop N Loop;

end Matrix Vector Multiply_Restricted;

end loop M Loop;

return Answer;

The following table describes the data objects maintained by this part:

```
_____
| Name | Type
                       Description
 Answer | Output_Vectors | Result of performing the matrix-vector | multiplication
3.3.6.2.10.2.5 PROCESS CONTROL
Not applicable.
3.3.6.2.10.2.6 PROCESSING
The following describes the processing performed by this part:
separate (General Vector Matrix Algebra)
function Matrix Vector Multiply Restricted
            (Matrix : Input Matrices;
            Vector : Input_Vectors) return Output_Vectors is

    --declaration section-

  Answer : Output Vectors;
--begin function Matrix_Vector_Multiply_Restricted
begin
  Answer := (others \Rightarrow 0.0);
  M Loop:
    for M in Indices1 loop
       N Loop:
          for N in Indices2 loop
             Answer(M) := Answer(M) +
```

Matrix(M, N) * Vector(N);





3.3.6.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.2.8 LIMITATIONS

None.

3.3.6.2.10.3 VECTOR_VECTOR_TRANSPOSE_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P444-0)

This function multiplies an m \times 1 input vector by the transpose of a n \times 1 input vector, returning the resultant m \times n matrix.

The following defines the result of this operation:

$$a(i,j) := b(i) * c(j)$$

3.3.6.2.10.3.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification for General Vector Matrix Algebra:

Data types:

The following table describes the generic formal types required by this part:



Name	Туре	Description
Left_Vector_Elements	floating point type	Data type of elements in left input vector
Right_Vector_Elements	floating point type	Data type of elements in right input vector
Matrix_Elements	floating point type	Data type of elements in output matrix
Indices1	discrete type	Used to dimension left input vector and first dimension of output matrix
Indices2	discrete type 	Used to dimension right input vector and second dimension of output matrix
Left Vectors	array	Data type of left input vector
Right Vectors	array	Data type of right input vector
Matrices	array	Data type of output matrix

The following table describes the generic formal subroutines required by this part:

1	Name		Туре	1	Description	I
	11 🛠 11		function		Operator defining the multiplication operation Left Vector_Elements * Right_Vector_Elements := Matrix_Elements	

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description	
Left	Left_Vectors	In	m x 1 vector	
Right	Right_Vectors	In	l x n vector	

3.3.6.2.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value D	Pescription
Answer	Matrices	N/A Re	sult matrix





```
3.3.6.2.10.3.5 PROCESS CONTROL
Not applicable.
3.3.6.2.10.3.6 PROCESSING
The following describes the processing performed by this part:
separate (General Vector Matrix Algebra)
function Vector_Vector_Transpose_Multiply_Restricted
            (Left : Left_Vectors ;
             Right: Right Vectors) return Matrices is
 -- --declaration section
   Answer : Matrices:
--begin function Vector_Vector_Transpose_Multiply_Restricted
begin
   M Loop:
     for M in Indices1 loop
         N Loop:
            for N in Indices2 loop
               Answer(M, N) := Left(M) * Right(N);
            end loop N Loop;
   end loop M Loop;
   return Answer;
end Vector_Vector_Transpose_Multiply_Restricted;
3.3.6.2.10.3.7 UTILIZATION OF OTHER ELEMENTS
None.
3.3.6.2.10.3.8 LIMITATIONS
None.
```

3.3.6.2.10.4 MATRIX_MATRIX_TRANSPOSE_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P447-0)

This function multiples an m \times n matrix by the transpose of a p \times n matrix, returning the resultant m \times p matrix.

The results of this operation are defined as follows:

$$a(i,j) := b(i,k) * c(j,k)$$

3.3.6.2.10.4.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.2.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described in the package specification for General Vector Matrix Algebra:

Data types:

The following table describes the generic formal types required by this part:

The following table describes the generic formal types required by this part:

Name	Type	Description
Left_Elements	floating point type	Type of elements in left input matrix
Right_Elements	floating point type	Type of elements in right input matrix
Output_Elements	floating point type	Type of elements in output matrix
M_Indices	discrete type	Used to dimension first dimension of left input matrix and output matrix
N_Indices	discrete type	Used to dimension second dimension of left and right input matrix
P_Indices	discrete type	Used to dimension first dimension of right input matrix and second dimension of output matrix
Left Matrices	array	Data type of left input matrix
Right Matrices	array	Data type of right input matrix
Output_Matrices	array	Data type of output matrix

.____



The following table describes the generic formal subroutines required by this part:

I	Name	 	Туре		Description	Ī
	U*11		function		Operator used to define the operation: Left_Elements * Right_Elements := Output_Elements	

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре			Mode		Description	-
Left Right	Left Right	Matrices Matrices	I	[n [n		Matrix to be used as the multiplicand Matrix whose transpose is to be used as the multiplier	

3.3.6.2.10.4.4 LOCAL DATA



Data objects:

The following table describes the data objects maintained by this part:

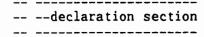
Type		Description	
	N/A	Result matrix being calculated	

3.3.6.2.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.4.6 PROCESSING

The following describes the processing performed by this part:



Answer : Output Matrices;



```
--begin function Matrix Matrix Transpose Multiply Restricted
begin
   Answer := (others \Rightarrow (others \Rightarrow 0.0));
   M Loop:
      for M in M_Indices loop
         P Loop:
            for P in P Indices loop
               N Loop:
                  for N in N Indices loop
                      Answer(M, P) := Answer(M, P) +
                                      Left(M, N) * Right(P, N);
                  end loop N Loop;
            end loop P Loop;
      end loop M_Loop;
   return Answer;
end Matrix_Matrix_Transpose_Multiply_Restricted;
3.3.6.2.10.4.7 UTILIZATION OF OTHER ELEMENTS
None.
3.3.6.2.10.4.8 LIMITATIONS
None.
3.3.6.2.10.5 DOT PRODUCT OPERATIONS RESTRICTED UNIT DESIGN (CATALOG #P450-0)
This function performs a dot product operation on two m-element vectors.
3.3.6.2.10.5.1 REQUIREMENTS ALLOCATION
This part meets CAMP requirement R063.
3.3.6.2.10.5.2 LOCAL ENTITIES DESIGN
None.
```



3.3.6.2.10.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined in the package specification for General_Vector_Matrix_Algebra.

Data types:

The following table describes the generic formal types required by this part:

_			
1	Name	Туре	Description
	Left_Elements	floating point type	Type of elements in left input vector
İ	Right_Elements	floating point type	Type of elements in right input vector
	Result_Elements	floating point type	Data type of result of dot product
	Indices	discrete type	Used to dimension input vectors
i	Left Vectors	array	Data type of left input vector
İ	Right_Vectors	array	Data type of right input vector

Subprograms:

The following table describes the generic formal subroutines required by this part:

1	Name	1	Туре		Description	
	n *u		function		Multiplication operator defining the operation: Left_Elements * Right_Elements := Result_Elements	1

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode	Description
Left	Left_Vectors	In	First vector in a dot product operation
Right	Right_Vectors	In	Second vector in a dot product operation

3.3.6.2.10.5.4 LOCAL DATA

Data objects:



The following table describes the data objects maintained by this part:

```
| Name | Type | Value | Description
| Answer | Result_Elements | N/A | Result of dot product operation
3.3.6.2.10.5.5 PROCESS CONTROL
Not applicable.
3.3.6.2.10.5.6 PROCESSING
The following describes the processing performed by this part:
separate (General Vector Matrix Algebra)
function Dot_Product_Operations_Restricted
           (Left : Left Vectors;
            Right: Right Vectors) return Result Elements is
-- --declaration section-
   Answer : Result Elements;
--begin function Dot_Product_Operations_Restricted
begin
   Answer := 0.0;
   Process:
      for Index in Indices loop
        Answer := Answer + Left(Index) * Right(Index);
     end loop Process;
   return Answer;
end Dot Product Operations Restricted;
3.3.6.2.10.5.7 UTILIZATION OF OTHER ELEMENTS
None.
3.3.6.2.10.5.8 LIMITATIONS
None.
```



3.3.6.2.10.6 DIAGONAL FULL MATRIX ADD RESTRICTED UNIT DESIGN (CATALOG #P453-0)

This function adds an m-element diagonal matrix to an m x m matrix, returning the resultant m x m matrix.

3.3.6.2.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R212.

3.3.6.2.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following table describes this part's generic parameters previously defined in the package specification for General_Vector_Matrix_Algebra:

Data types:

The following table describes the generic formal types required by this part:



Name	Туре	Description
Elements	floating point type	Type of elements in input and output arrays
Diagonal_Range	integer type	Used to dimension Diagonal Matrices
Indices	discrete type	Used to dimension input and output matrices
Diagonal Matrices	array	Data type of diagonal input matrix
Full_Matrices	array	Data type of full input and output matrices

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode	Description	1
D_Matrix F_Matrix	Diagonal_Matrices Full_Input_Matrices	In In	Input diagonal matrix Input full matrix to be added to the diagonal matrix	,



3.3.6.2.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value Description	
Answer Diag_Index	Full_Output_Matrices Diagonal_Range	N/A Resultant matrix N/A Index into diagonal matrix	
Index	Indices	N/A Index into full matrix	ж

3.3.6.2.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.2.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
separate (General Vector Matrix Algebra)
function Diagonal Full Matrix Add Restricted
(D Matrix: Diagonal Matrices;
```

F Matrix : Full Matrices) return Full Matrices is

```
-- --declaration section-
```

Answer : Full_Matrices;
Diag_Index : Diagonal_Range;

Index : Indices;

--begin function Diagonal_Full_Matrix_Add_Restricted

begin

```
-- -- assign all values to answer and then add in diagonal elements
Answer := F Matrix;
```

-- -- now add in diagonal elements

```
Diag_Index := Diagonal Range'FIRST;
Index := Indices'FIRST;
```

Add Loop: Ioop

```
Answer(Index, Index) := Answer(Index, Index) + D_Matrix(Diag_Index);
exit when Index = Indices'LAST;
```



Diag_Index := Diagonal_Range'SUCC(Diag_Index);
Index := Indices'SUCC(Index);

end loop Add Loop;

return Answer;

end Diagonal Full Matrix Add Restricted;

3.3.6.2.10.6.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.2.10.6.8 LIMITATIONS

None.

3.3.6.2.10.7 VECTOR_MATRIX_MULTIPLY_RESTRICTED UNIT DESIGN (CATALOG #P438-0)

This package contains a function which multiplies a $1 \times m$ vector by an $m \times n$ matrix producing a $1 \times n$ vector.

The calculations performed are as follows:

$$c(j) := a(i) * b(i,j)$$

The function can be made to handle sparse matrices and/or vectors by tailoring the imported "+" and "*" functions (see sections describing generic formal subprograms and calling sequence).

3.3.6.2.10.7.1 REQUIREMENTS ALLOCATION

N/A

•

3.3.6.2.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.2.10.7.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of General_Vector_Matrix_Algebra.

Data types:

The following table describes the generic formal types required by this part:



Name	Туре	Description
Matrix_Elements	floating point type	Type of elements in the input matrix
Input_Vector_Elements	floating point type	Type of elements in the input vector
Output_Vector_Elements	floating point type	Type of elements in the output vector
Indices1	discrete type	Used to dimension first dimension of input matrix and to dimension
Indices2	discrete type	the output vector Used to dimension second dimension of input matrix and to dimension the input vector
Input Matrices	array	Data type of input matrix
Input Vectors	array	Data type of input vector
Output_Vectors	array	Data type of output vector

Subprograms:

The following table describes the generic formal subroutines required by this part. This function can be made to handle sparse matrices and/or vectors by tailoring the imported functions to check the appropriate element(s) for zero before performing the indicated operation.

Name	Туре	Description	1
"*"	function	Function defining the operation Input_Vector_Elements * Matrix_Elements := Output Vector Elements	
"+" 	function	Function defining the operation Output Vector Elements + Output Vector Elements := Output Vector Elements	

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name	I	Туре		Mode	1	Description	- -
	Vector Matrix		Input_Vectors Input_Matrices		In In		M-element vector M x N input matrix	

3.3.6.2.10.7.4 LOCAL DATA

Data objects:

(

None.

```
The following table describes the data objects maintained by this part:
| Name | Type | Value | Description
| Answer | Output_Vectors | N/A | Vector being calculated and returned
3.3.6.2.10.7.5 PROCESS CONTROL
Not applicable.
3.3.6.2.10.7.6 PROCESSING
The following describes the processing performed by this part:
separate (General Vector Matrix Algebra)
function Vector Matrix Multiply Restricted
           (Vector: Input Vectors;
            Matrix: Input Matrices) return Output Vectors is
-- --declaration section
  Answer : Output_Vectors := (others => 0.0);
--begin function Vector Matrix Multiply Restricted
______
begin
  N Loop:
     for N in Indices2 loop
        M Loop:
           for M in Indices1 loop
              Answer(N) := Answer(N) + Vector(M) * Matrix(M,N);
           end loop M_Loop;
     end loop N Loop;
  return Answer;
end Vector Matrix Multiply Restricted;
3.3.6.2.10.7.7 UTILIZATION OF OTHER ELEMENTS
```

3.3.6.2.10.7.8 LIMITATIONS

None.

```
. 193
```

```
package body General Vector Matrix Algebra is
  package body Vector Operations Unconstrained is separate;
  package body Vector Operations Constrained is separate;
  package body Matrix Operations Unconstrained is separate;
  package body Matrix Operations Constrained is separate;
  package body Dynamically Sparse Matrix_Operations_Unconstrained is separate;
  package body Dynamically Sparse Matrix Operations Constrained is separate;
  package body Symmetric Half Storage Matrix Operations is separate;
  package body Symmetric Full Storage Matrix Operations Unconstrained is separate;
  package body Symmetric Full Storage Matrix Operations Constrained is separate;
  package body Diagonal Matrix Operations is separate;
  package body Vector Scalar Operations Unconstrained is separate;
  package body Vector Scalar Operations Constrained is separate;
  package body Matrix Scalar Operations Unconstrained is separate;
  package body Matrix Scalar Operations Constrained is separate;
  package body Diagonal Matrix Scalar Operations is separate;
  package body Matrix Vector Multiply Unrestricted is separate;
  function Matrix Vector Multiply Restricted
              (Matrix : Input Matrices;
               Vector : Input_Vectors) return Output_Vectors is separate;
  package body Vector Vector Transpose Multiply Unrestricted is separate;
  function Vector Vector Transpose Multiply Restricted
              (Left : Left Vectors;
               Right: Right Vectors) return Matrices is separate;
  package body Matrix Matrix Multiply Unrestricted is separate;
  function Matrix Matrix Multiply Restricted
              (Left : Left Matrices;
               Right: Right Matrices) return Output Matrices is separate;
  package body Matrix Matrix Transpose Multiply Unrestricted is separate;
  function Matrix Matrix Transpose Multiply Restricted
              (Left : Left Matrices;
               Right: Right Matrices) return Output Matrices is separate;
  package body Dot Product Operations Unrestricted is separate;
```

```
function Dot Product Operations Restricted
               (Left : Left Vectors;
                Right : Right Vectors)
               return Result Elements is separate;
   package body Diagonal Full Matrix Add Unrestricted is separate;
   function Diagonal Full Matrix Add_Restricted
               (D Matrix : Diagonal Matrices;
                F Matrix : Full Matrices) return Full Matrices is separate;
   package body Vector Matrix Multiply Unrestricted is separate;
   function Vector Matrix Multiply Restricted
               (Vector : Input Vectors;
                Matrix : Input Matrices) return Output Vectors is separate;
   package body Aba Trans Dynam Sparse Matrix Sq Matrix is separate;
   package body Aba Trans Vector Sq Matrix is separate;
   package body Aba Trans Vector Scalar is separate;
   package body Column Matrix_Operations is separate;
end General Vector Matrix Algebra;
```

```
separate (General Vector Matrix Algebra)
package body Vector Operations Unconstrained is
pragma PAGE;
   function "+" (Left : Vectors;
                 Right: Vectors) return Vectors is
     -- declaration section-
      Answer : Vectors(Left'range);
      L Index : Indices;
      R Index : Indices;
-- -- begin function "+"
__ _____
   begin
      -- make sure lengths of input vectors are the same
      if Left'LENGTH = Right'LENGTH then
         L Index := Left'FIRST;
         R Index := Right'FIRST;
         Process:
            loop
               Answer(L Index) := Left(L_Index) + Right(R_Index);
               exit Process when L_Index = Left'LAST;
               L Index := Indices'SUCC(L Index);
               R Index := Indices'SUCC(R Index);
            end loop Process;
      else
         -- dimensions of vectors are incompatible
         raise Dimension Error;
      end if;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Left : Vectors;
                 Right: Vectors) return Vectors is
      -- declaration section-
```

```
Answer : Vectors(Left'range);
L_Index : Indices;
      R Index : Indices;
-- -- begin function "-"
__ _____
   begin
      -- make sure lengths of the input vectors are the same
      if Left'LENGTH = Right'LENGTH then
         L Index := Left'FIRST;
         R Index := Right'FIRST;
         Process:
             loop
                Answer(L Index) := Left(L Index) - Right(R Index);
                exit Process when L Index = Left'LAST;
                L Index := Indices'SUCC(L Index);
                R Index := Indices'SUCC(R Index);
            end loop Process;
      else
         -- dimensions of vectors are incompatible
         raise Dimension Error;
      end if;
      return Answer;
   end "-";
pragma PAGE;
   function Vector Length (Input: Vectors) return Vector Elements is
      -- declaration section-
      ______
      Temp
             : Vector Elements Squared;
-- -- begin function Vector Length
   begin
      Temp := 0.0;
```

```
Process:
         for INDEX in Input'range loop
            Temp := Temp +
                    Input(INDEX) * Input(INDEX);
         end loop Process;
      return Sqrt(Temp);
   end Vector Length;
pragma PAGE;
   function Dot Product (Left : Vectors;
                         Right: Vectors) return Vector_Elements_Squared is
     -- declaration section-
      -----
     Answer : Vector Elements Squared;
     L_Index : Indices;
     R Index : Indices;
-- -- begin function Dot Product
  begin
      -- make sure lengths of the input vectors are the same
      if Left'LENGTH = Right'LENGTH then
         Answer := 0.0;
         L Index := Left'FIRST;
         R Index := Right'FIRST;
         Process:
            loop
               Answer := Answer + Left(L Index) * Right(R Index);
               exit Process when L_Index = Left'LAST;
               L Index := Indices'SUCC(L Index);
               R_Index := Indices'SUCC(R_Index);
           end loop Process;
     else
         -- dimensions of vectors are incompatible
        raise Dimension Error;
     end if;
     return Answer;
  end Dot_Product;
```

end Vector_Operations_Unconstrained;

```
100
```

```
separate (General Vector Matrix Algebra)
package body Matrix Operations Unconstrained is
pragma PAGE;
   function "+" (Left : Matrices;
                  Right: Matrices) return Matrices is
      -- declaration section-
      Answer : Matrices(Left'range(1), Left'range(2));
      L_Col : Col_Indices;
L_Row : Row_Indices;
R_Col : Col_Indices;
R_Row : Row_Indices;
__ _____
-- -- begin function "+"
   begin
      -- make sure the dimensions of the matrices are compatible
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then
         L Row := Left'FIRST(1);
         R Row := Right'FIRST(1);
         Row Loop:
             IOOD
                L Col := Left'FIRST(2);
                R Col := Right'FIRST(2);
                Col Loop:
                   Toop
                       Answer(L_Row, L_Col) := Left(L_Row, L_Col) +
                                                  Right(\overline{R}_Row, \overline{R} Col);
                       exit Col Loop when L Col = Left'LAST(2);
                       L Col := Col Indices SUCC(L Col);
                       R Col := Col Indices'SUCC(R Col);
                   end loop Col Loop;
                exit Row Loop when L Row = Left'LAST(1);
                L Row := Row Indices'SUCC(L Row);
                R Row := Row Indices'SUCC(R Row);
             end loop Row Loop;
      else
         -- input matrices have incompatible dimensions
         raise Dimension Error;
```

```
end if;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Left : Matrices;
                 Right: Matrices) return Matrices is
      -----
      -- declaration section-
      ------
      Answer : Matrices(Left'range(1), Left'range(2));
     L_Col : Col_Indices;
L_Row : Row_Indices;
R_Col : Col_Indices;
R_Row : Row_Indices;
-- -----
-- -- begin function "-"
-- -----
  begin
      -- make sure matrix dimensions are compatible
     if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then
         L Row := Left'FIRST(1);
         R Row := Right'FIRST(1);
         Row Loop:
            Гоор
               L Col := Left'FIRST(2);
               R Col := Right'FIRST(2);
               Col Loop:
                  Toop
                     Answer(L_Row, L_Col) := Left(L_Row, L_Col) -
                                              Right(R_Row, R_Col);
                     exit Col Loop when L Col = Left'LAST(2);
                     L Col := Col Indices SUCC(L Col);
                     R Col := Col Indices'SUCC(R_Col);
                  end loop Col Loop;
               exit Row Loop when L Row = Left'LAST(1);
               L Row := Row Indices SUCC(L Row);
               R Row := Row Indices'SUCC(R Row);
            end loop Row Loop;
     else
```

```
-- input matrices have incompatible dimensions
         raise Dimension Error;
      end if;
      return Answer;
   end "-";
pragma PAGE;
   function "+" (Matrix : Matrices;
                 Addend : Elements) return Matrices is
      -- declaration section-
      Answer: Matrices(Matrix'range(1), Matrix'range(2));
-- -- begin function "+"
   begin
      Row Loop:
         for Row in Matrix'range(1) loop
            Col Loop:
               for COL in Matrix'range(2) loop
                  Answer(Row, COL) := Matrix(Row, COL) + Addend;
               end loop Col Loop;
         end loop Row Loop;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Matrix : Matrices;
                 Subtrahend: Elements) return Matrices is
      -- declaration section-
      Answer: Matrices(Matrix'range(1), Matrix'range(2));
-- -- begin function "-"
__ ____
   begin
      Row Loop:
         for Row in Matrix'range(1) loop
            Col Loop:
               for COL in Matrix'range(2) loop
```

```
Answer(Row, COL) := Matrix(Row, COL) - Subtrahend;
                end loop Col Loop;
         end loop Row Loop;
      return Answer;
   end "-";
pragma PAGE;
   procedure Set To Identity Matrix (Matrix: out Matrices) is
      -- declaration section
      _____
      Col Marker : Col Indices;
      Row : Row Indices;
-- -- begin function Set To Identity Matrix
   begin
      -- make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
         Matrix := (others => (others => 0.0));
                    := Matrix'FIRST(1);
         Col Marker := Matrix'FIRST(2);
         Row Loop:
            Toop
                -- set diagonal element equal to 1
               Matrix(Row, Col Marker) := 1.0;
                exit Row Loop when Row = Matrix'LAST(1);
                         := Row Indices'SUCC(Row);
                Col Marker := Col Indices'SUCC(Col Marker);
            end loop Row Loop;
      else
         -- do not have a square matrix
         raise Dimension Error;
      end if:
   end Set_To_Identity_Matrix;
pragma PAGE;
   procedure Set To Zero Matrix (Matrix: out Matrices) is
   begin
```

```
Matrix := (others => (others => 0.0));
   end Set To Zero Matrix;
pragma PAGE;
   function "*" (Left : Matrices;
                 Right: Matrices) return Matrices is
     _____
     -- declaration section
     _____
     Answer : Matrices(Left'range(1), Right'range(2));
            : Row Indices;
     N Left : Col Indices;
     N_Right : Row_Indices;
            : Col Indices;
-- -- begin function "*"
  -----
  begin
     -- make sure dimensions are compatible
     if Left'LENGTH(2) = Right'LENGTH(1) then
        M := Left'FIRST(1);
        M Loop:
           loop
              P := Right'FIRST(2);
              P Loop:
                 loop
                    Answer(M,P) := 0.0;
                    N Left := Left'FIRST(2);
                    N Right := Right'FIRST(1);
                    N Loop:
                       loop
                          Answer(M,P) := Answer(M,P) +
                                         Left(M,N Left) * Right(N Right,P);
                          exit N Loop when N Left = Left'LAST(2);
                          N_Left := Col_Indices'SUCC(N_Left);
                          N_Right := Row_Indices'SUCC(N_Right);
                       end loop N Loop;
                    exit P Loop when P = Right'LAST(2);
                    P := Col Indices'SUCC(P);
                 end loop P Loop;
              exit M Loop when M = Left'LAST(1);
```

```
sepreste (General Vector Matrix Algebra)
package body Dynamically Sparse Matrix Operations Unconstrained is
pragma PAGE;
   procedure Set To Identity Matrix (Matrix: out Matrices) is
      -- declaration section
      Col Marker : Col Indices;
      Row : Row Indices;
-- -- begin procedure Set to Identity Matrix
   begin
      -- make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
         Matrix := (others => (others => 0.0));
                     := Matrix'FIRST(1);
         Col Marker := Matrix'FIRST(2);
         Row Loop:
             Toop
                -- set diagonal element equal to 1.0
                Matrix(Row, Col Marker) := 1.0;
                exit Row Loop when Row = Matrix'LAST(1);
                          := Rov_Indices'SUCC(Row);
                Col Marker := Col Indices'SUCC(Col Marker);
             end loop Row Loop;
      else
         raise Dimension Error;
      end if;
   end Set_To_Identity_Matrix;
pragma PAGE;
   procedure Set To Zero Matrix (Matrix: out Matrices) is
   begin
      Matrix := (others => (others => 0.0));
   end Set To Zero Matrix;
pragma PAGE;
   function Add To Identity (Input: Matrices) return Matrices is
```

```
_____
     -- declaration section
      _____
     Answer : Matrices(Input'range(1), Input'range(2));
     Col Marker : Col Indices;
     Row : Row Indices;
-- -- begin procedure Add to Identity
  begin
      -- make sure input is a square matrix
     if Input'LENGTH(1) = Input'LENGTH(2) then
        Answer := Input;
         -- add "identity" values to diagonal elements
        Row := Input'FIRST(1);
        Col Marker := Input'FIRST(2);
        Row Loop:
            loop
               if Answer(Row, Col Marker) /= 0.0 then
                  Answer(Row, Col_Marker) := Answer(Row, Col_Marker) + 1.0;
               else
                  Answer(Row, Col_Marker) := 1.0;
               end if;
               exit Row Loop when Row = Input'LAST(1);
               Row := Row Indices'SUCC(Row);
               Col Marker := Col Indices'SUCC(Col Marker);
            end loop Row Loop;
     else
        raise Dimension_Error;
     end if;
     return Answer;
  end Add To Identity;
pragma PAGE;
  function Subtract From Identity (Input: Matrices) return Matrices is
     -- declaration section
     Answer : Matrices(Input'range(1),Input'range(2));
     Col Marker : Col Indices;
```

```
Row
                 : Row Indices;
-- -- begin procedure Subtract From Identity
   begin
      -- make sure input is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then
                     := Input'FIRST(1);
         Col Marker := Input'FIRST(2);
         Row Loop:
             loop
                Col Loop:
                   for COL in Input'range(2) loop
                      if Input(Row, COL) /= 0.0 then
                         Answer(Row,COL) := - Input(Row,COL);
                         Answer(Row, COL) := 0.0;
                      end if;
                   end loop Col Loop;
                if Answer(Row, Col Marker) /= 0.0 then
                   Answer(Row, Col Marker) := Answer(Row, Col Marker) + 1.0;
                   Answer(Row, Col Marker) := 1.0;
                end if;
                exit Row Loop when Row = Input'LAST(1);
                          := Row Indices'SUCC(Row);
                Col_Marker := Col_Indices'SUCC(Col_Marker);
            end loop Row Loop;
      else
         raise Dimension Error;
      end if;
      return Answer;
   end Subtract From Identity;
pragma PAGE;
   function "+" (Left : Matrices;
                 Right: Matrices) return Matrices is
      -- declaration section
      Answer: Matrices(Left'range(1), Left'range(2));
      L Col : Col Indices;
```

```
L Row : Row Indices;
      R Col : Col Indices:
      R Row : Row Indices;
-- -- begin function "+"
__ _____
   begin
      -- make sure have compatible dimensions
      if Left'LENGTH(1) = Right'LENGTH(1) and then
         Left'LENGTH(2) = Right'LENGTH(2) then
         L Row := Left'FIRST(1);
         R Row := Right'FIRST(1);
         Row Loop:
            Toop
                L Col := Left'FIRST(2);
                R Col := Right'FIRST(2);
                Col Loop:
                   Ioop
                      if Left(L_Row, L_Col) = 0.0 then
                         if Right (R Row, R Col) = 0.0 then
                            Answer(\overline{L}Row, \overline{L}Col) := 0.0;
                            Answer(L_Row, L_Col) := Right(R_Row, R_Col);
                         end if:
                      elsif Right(R Row, R Col) = 0.0 then
                         Answer(L Row, L Col) := Left(L Row, L Col);
                         Answer(L Row, L Col) := Left(L Row, L Col) +
                                                   Right(\overline{R}_{Row}, \overline{R}_{Col});
                      end if;
                      exit Col Loop when L Col = Left'LAST(2);
                      L_Col := Col_Indices'SUCC(L_Col);
                      R Col := Col Indices'SUCC(R Col);
                   end loop Col Loop;
               exit Row Loop when L Row = Left'LAST(1);
               L Row := Row Indices'SUCC(L Row);
               R Row := Row Indices'SUCC(R Row);
            end loop Row Loop;
     else
         raise Dimension Error;
     end if;
     return Answer;
```

```
end "+";
pragma PAGE;
   function "-" (Left : Matrices;
                 Right: Matrices) return Matrices is
      -- declaration section
     Answer: Matrices(Left'range(1), Left'range(2));
     L Col : Col Indices;
      L Row : Row Indices:
     R Col : Col Indices;
     R Row : Row Indices;
-- -- begin function "-"
__ _____
  begin
      -- make sure have compatible dimensions
      if Left'LENGTH(1) = Right'LENGTH(1) and
         Left'LENGTH(2) = Right'LENGTH(2) then
         L Row := Left'FIRST(1);
         R Row := Right'FIRST(1);
         Row Loop:
            Toop
               L Col := Left'FIRST(2);
               R Col := Right'FIRST(2);
               Col Loop:
                  Ioop
                      if Left(L_Row, L_Col) = 0.0 then
                         if Right (R_Row, R Col) = 0.0 then
                            Answer(\bar{L} Row, \bar{L} Col) := 0.0;
                            Answer(L Row, L Col) := - Right(R Row, R Col);
                         end if;
                      elsif Right(R Row, R Col) = 0.0 then
                         Answer(L_Row, L_Col) := Left(L_Row, L_Col);
                         Answer(L Row, L Col) := Left(L Row, L Col) -
                                                  Right(\overline{R} Row, \overline{R} Col);
                     end if;
                     exit Col Loop when L Col = Left'LAST(2);
                      L_Col := Col Indices SUCC(L Col);
                     R Col := Col Indices'SUCC(R_Col);
                  end loop Col Loop;
               exit Row Loop when L Row = Left'LAST(1);
               L Row := Row Indices'SUCC(L Row);
```

343

```
R_Row := Row_Indices'SUCC(R_Row);
end loop Row_Loop;
else
    raise Dimension_Error;
end if;
return Answer;
end "-";
end Dynamically_Sparse_Matrix_Operations_Unconstrained;
```

```
separate (General Vector Matrix Algebra)
package body Symmetric Half Storage Matrix Operations is
-- -- local declarations
   type Col Index Arrays is array(Col Indices) of NATURAL;
   type Row Index_Arrays is array(Row_Indices) of NATURAL;
   Col Offset : Col Index Arrays;
   Row Marker : Row Index Arrays;
-- -- this object is initially only zeroed out; the 1.0 values will be assigned
-- -- to the diagonal elements during package initialization
   Local Identity Matrix : Matrices := (others => 0.0);
   Local Zero Matrix : constant Matrices := (others => 0.0);
pragma PAGE;
   function Swap_Col (Row: Row_Indices) return Col Indices is
      return Col Indices'VAL(Row Indices'POS(Row) -
                                Row Indices'POS(Row Indices'FIRST) +
                                Col Indices'POS(Col Indices'FIRST));
   end Swap_Col;
pragma PAGE;
   function Swap Row (COL: Col Indices) return Row Indices is
      return Row Indices'VAL(Col Indices'POS(COL) -
                                Col Indices'POS(Col Indices'FIRST) +
                                Row Indices'POS(Row Indices'FIRST));
   end Swap_Row;
pragma PAGE;
   procedure Initialize (Row_Slice : in Row_Slices; Row : in Row_Indices;
                           Row: in
                           Matrix : out Matrices) is
      -- declaration section
      INDEX : Col_Indices;
Marker : POSITIVE;
      Stop Here : POSITIVE;
-- -- begin procedure Initialize
   begin
      INDEX := Col_Indices'FIRST;
Marker := Row Marker(Row);
      Stop_Here := Marker + Col_Offset(Swap Col(Row));
```

```
Process:
         loop
            Matrix(Marker) := Row Slice(INDEX);
            exit Process when Marker = Stop Here;
            INDEX := Col Indices'SUCC(INDEX);
            Marker := Marker + 1:
         end loop Process;
   end Initialize;
pragma PAGE;
   function Identity Matrix return Matrices is
   begin
      return Local Identity Matrix;
   end Identity Matrix;
pragma PAGE;
   function Zero Matrix return Matrices is
   begin
      return Local Zero Matrix;
   end Zero Matrix;
pragma PAGE;
   procedure Change Element (New Value : in
                                                  Elements;
                               Row
                                       : in
                                                  Row Indices:
                                       : in
                                                  Col Indices;
                               COL
                               Matrix
                                        : out Matrices) is
   begin
      -- determine which half of the matrix is being referenced
      if Row_Indices'POS(Row) - Row_Indices'POS(Row_Indices'FIRST) >=
         Col Indices' POS(COL) - Col Indices' POS(Col Indices' FIRST) then
         -- looking at bottom half of array
         Matrix(Row_Marker(Row) + Col_Offset(COL)) := New_Value;
      else
         -- looking at top half; need to switch to bottom half
         Matrix(Row Marker(Swap Row(COL)) +
                Col Offset(Swap Col(Row))) := New Value;
      end if;
   end Change Element;
pragma PAGE;
```

```
function Retrieve Element (Matrix: Matrices;
                                  Row
                                          : Row_Indices;
                                          : Col Indices) return Elements is
       -- declaration section
      Answer : Elements;
-- -- begin function Retrieve Element
   begin
      -- determine which half of the array is being referenced
      if Row_Indices'POS(Row) - Row_Indices'POS(Row Indices'FIRST) >=
         Col_Indices'POS(COL) - Col_Indices'POS(Col_Indices'FIRST) then
         -- already looking at the bottom half of the array
         Answer := Matrix(Row Marker(Row) + Col Offset(COL));
     else
         -- looking at the top half; need to switch to bottom half
        Answer := Matrix(Row Marker(Swap Row(COL)) +
                            Col_Offset(Swap_Col(Row)));
     end if;
     return Answer;
   end Retrieve Element;
pragma PAGE;
   function Row_Slice (Matrix : Matrices;
                         Row
                                : Row Indices) return Row Slices is
      -- declaration section
      Answer : Row_Slices;
-- -- begin function Row Slice
   begin
      -- retrieve row elements in bottom half of array
      Bottom Loop:
          for COL in Col Indices'FIRST .. Swap Col(Row) loop
             Answer(COL) := Matrix(Row Marker(Row) + Col_Offset(COL));
          end loop Bottom Loop;
```

```
-- retrieve row elements in top half of array, if there are any
       if Row /= Row Indices'LAST then
          Top Loop:
             for COL in Col Indices'SUCC(Swap Col(Row)) .. Col_Indices'LAST loop
                Answer(COL) := Matrix(Row Marker(Swap Row(COL)) +
                                         Col Offset(Swap Col(Row)));
             end loop Top Loop;
       end if;
       return Answer;
   end Row Slice;
pragma PAGE;
   function Column_Slice (Matrix : Matrices;
                             COL
                                 : Col Indices) return Col Slices is
      -- declaration section
      Answer : Col_Slices;
-- -- begin function Column Slice
   begin
       -- retrieve column elements contained in bottom half of array
      Bottom Loop:
          for Row in Swap Row(COL) .. Row Indices'LAST loop
             Answer(Row) := Matrix(Row Marker(Row) + Col Offset(COL));
          end loop Bottom Loop;
      -- retrieve column elements contained in top half of array, if any
      if COL /= Col Indices'FIRST then
          Top Loop:
             for Row in Row_Indices'FIRST .. Row_Indices'PRED(Swap_Row(COL)) loop
                Answer(Row) := Matrix(Row Marker(Swap Row(COL)) +
                                         Col Offset(Swap Col(Row)));
             end loop Top_Loop;
      end if;
      return Answer;
   end Column Slice;
pragma PAGE;
   function Add To Identity (Input: Matrices) return Matrices is
      -- declaration section
      Answer : Matrices;
```

```
-- -- begin function Add To Identity
   begin
       -- do straight assignment of all elements and then add in the
       -- identity matrix
       Answer := Input;
       -- all diagonal elements, except for the last one, are located one
       -- entry before the starting location of the next row
       Add Identity Loop:
          for INDEX in Row Indices' SUCC(Row Indices' FIRST) ..
                          Row Indices'LAST loop
              Answer(Row Marker(INDEX) - 1) := Answer(Row Marker(INDEX)-1) + 1.0;
          end loop Add Identity_Loop;
       -- handle last diagonal element
       Answer(Entry Count) := Answer(Entry Count) + 1.0;
       return Answer;
   end Add To Identity;
pragma PAGE;
   function Subtract_From_Identity (Input : Matrices) return Matrices is
       -- declaration section
       Answer : Matrices;
  -- begin function Subtract from Identity
   begin
       -- subtract Input from a zero matrix and then add it to an identity matrix
      Subtract Loop:
          for INDEX in 1.. Entry_Count loop
              Answer(INDEX) := - Input(INDEX);
          end loop Subtract Loop;
      -- all diagonal elements, except for the last one, are located one
      -- entry before the starting location of the next row
      Add Identity Loop:
          for INDEX in Row Indices' SUCC(Row Indices' FIRST) ..
                         Row Indices'LAST loop
                          Marker(INDEX) - 1) := Answer(Row Marker(INDEX)-1) + 1.0;
              Answer(Ru
          end loop Add Identity Loop;
      -- handle last diagonal element
```

```
Answer(Entry Count) := Answer(Entry Count) + 1.0;
      return Answer;
   end Subtract From Identity;
pragma PAGE;
   function "+" (Left : Matrices;
                 Right: Matrices) return Matrices is
      -- declaration section
__
      Answer : Matrices;
-- -- begin function "+"
-- -----
   begin
      Process:
         for INDEX in 1.. Entry Count loop
            Answer(INDEX) := Left(INDEX) + Right(INDEX);
         end loop Process;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Left : Matrices;
                 Right: Matrices) return Matrices is
      -- declaration section
      Answer : Matrices;
-- -- begin function "-"
   begin
      Process:
         for INDEX in 1 .. Entry_Count loop
            Answer(INDEX) := Left(INDEX) - Right(INDEX);
         end loop Process;
      return Answer;
   end "-";
pragma PAGE;
```

```
-- begin processing for Symmetric Half Storage
-- Matrix Operations package body
begin
   Init Block:
       declare
          COUNT
                               : NATURAL;
          Offset
                               : NATURAL;
          Row Starting Point : NATURAL;
       begin
          -- make sure lengths of row and col indices are the same
          if Row Slices'LENGTH /= Col Slices'LENGTH then
              raise Dimension Error;
          else
              -- initialize row marker identity matrix arrays;
              -- all diagonal elements, except for the last one, which require
              -- a value of 1 for the identity matrix are located one entry
                  before the starting location of the next row
              _____
              -- handle first row marker entry to simplify initialization of
              -- the identity matrix -- (NOTE: count implicitly equals 0)
              Row Marker(Row Indices'FIRST) := 1;
              COUNT := 1;
              Row Marker And Identity Matrix Init Loop:
                 for INDEX in Row Indices' SUCC(Row Indices' FIRST) ...
                                Row Indices' LAST loop
                    Row Starting Point := (COUNT * (COUNT+1) / 2) + 1;
                    Row Marker(INDEX) := Row Starting Point;
                    Local Identity Matrix(Row_Starting_Point-1) := 1.0;
                    COUNT := COUNT + 1;
                 end loop Row Marker And Identity Matrix Init Loop;
              -- initialize last diagonal element
              Local Identity Matrix(Entry Count) := 1.0;
              -- initialize column offset array
             Offset := 0;
             Col Marker Init Loop:
```

```
for INDEX in Col Indices loop
        Col_Offset(INDEX) := Offset;
        Offset := Offset + 1;
    end loop Col_Marker_Init_Loop;
end if;
```

end Init Block;

end Symmetric_Half_Storage_Matrix_Operations;

```
separate (General_Vector_Matrix_Algebra)
package body Symmetric Full Storage Matrix Operations Unconstrained is
pragma PAGE;
   procedure Change Element (New Value : in
                                                  Elements;
                              Row : in
                                                  Row Indices;
                              COL
                                       : in
                                                  Col Indices;
                              Matrix : in out Matrices) is
      -- declaration section-
      -----
      S Col : Col Indices;
      S Row : Row Indices;
-- -- begin procedure Change Element-
   begin
      -- make sure you have a square matrix
      if Matrix'LENGTH(1) /= Matrix'LENGTH(2) then
         raise Dimension Error;
      -- make sure row and col are within bounds
      elsif not (Row in Matrix'range(1) and
                  COL in Matrix'range(2)) then
         raise Invalid Index;
      else
         -- everything is okay
         S Col := Col Indices' VAL(Row Indices' POS(Row) -
                                   Row Indices'POS(Matrix'FIRST(1)) +
                                   Col Indices'POS(Matrix'FIRST(2)));
         S Row := Row Indices'VAL(Col Indices'POS(COL) -
                                   Col Indices'POS(Matrix'FIRST(2)) +
                                   Row Indices'POS(Matrix'FIRST(1)));
         Matrix(Row, COL)
                            := New Value;
         Matrix(S Row, S Col) := New Value;
      end if;
   end Change Element;
pragma PAGE;
   procedure Set_To_Identity_Matrix (Matrix : out Matrices) is
-- -- declaration section-
```

```
__ _____
   COL : Col Indices;
   Row : Row Indices;
-- -- begin procedure Set to Identity-
-- -----
   begin
      -- make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
         Matrix := (others => (others => 0.0));
         Row := Matrix'FIRST(1);
         COL := Matrix'FIRST(2);
         Row Loop:
            Toop
               -- set diagonal element equal to
               Matrix(Row, COL) := 1.0;
               exit Row Loop when Row = Matrix'LAST(1);
               Row := Row Indices'SUCC(Row);
               COL := Col Indices'SUCC(COL);
            end loop Row Loop;
      else
         -- do not have a square matrix
         raise Dimension Error;
      end if:
  end Set To Identity Matrix;
pragma PAGE;
  procedure Set To Zeio Matrix (Matrix: out Matrices) is
  begin
     Matrix := (others => (others => 0.0));
  end Set To Zero Matrix;
pragma PAGE;
  function Add_To_Identity (Input : Matrices) return Matrices is
     -- declaration section
     _____
     Answer : Matrices(Input'range(1), Input'range(2));
```

()

```
COL
                  : Col Indices;
                 : Row Indices;
      Row
-- -- begin function Add to Identity
-- -----
   begin
      -- make sure input matrix is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then
         Answer := Input;
         Row := Input'FIRST(1);
         COL := Input'FIRST(2);
         Access Diagonal Elements:
            loop
               Answer(Row, COL) := Answer(Row, COL) + 1.0;
               exit Access Diagonal Elements when Row = Input'LAST(1);
               Row := Row Indices'SUCC(Row);
               COL := Col Indices'SUCC(COL);
            end loop Access Diagonal Elements;
      else
         -- do not have a square matrix
         raise Dimension Error;
      end if;
      return Answer;
   end Add To Identity;
pragma PAGE;
   function Subtract From Identity (Input: Matrices) return Matrices is
      -- declaration section
     Answer : Matrices(Input'range(1), Input'range(2));
COL : Col_Indices;
     Col Count : POSITIVE;
     Row : Row Indices;
     Row Count : POSTTIVE;
     S_Col : Col_Indices;
      S Row
               : Row Indices;
  -- begin function Subtract from Identity
```

begin

```
-- make sure input matrix is a square matrix
if Input'LENGTH(1) = Input'LENGTH(2) then
   -- will subtract input matrix from an identity matrix by first
   -- subtracting all elements from 0.0 and then adding 1.0 to the
   -- diagonal elements;
   -- when doing the subtraction, will only calculate the remainder
   -- for the elements in the bottom half of the matrix and will simply
   -- do assignments for the symmetric elements in the top half of the
   -- matrix
   Row Count := 1;
   -- S Col will go across the columns as Row goes down the rows;
   -- will mark column containing the diagonal element for this row
   Row := Input'FIRST(1);
   S Col := Input'FIRST(2);
   Do Every Row:
       loop
           Col Count := 1;
           -- S Row will go down the rows as Col goes across the columns;
           -- when paired with S Col will mark the symmetric counterpart
           -- to the element being referenced in the bottom half of the
           -- matrix
           COL
                       := Input'FIRST(2);
                       := Input'FIRST(1);
           Subtract Elements From Zero:
               loop
                  -- perform subtraction on element in bottom half of matrix
                  Answer(Row, COL) := - Input(Row, COL);
                  -- exit loop after diagonal element has been reached
                  exit Subtract Elements From Zero when Col Count =
                                                                     Row Count;
                  -- assign values to symmetric elements in top half of matrix
                  -- (done after check for diagonal, since diagonal elements
                  -- don't have a symmetric counterpart)
                  Answer(S Row, S Col) := Answer(Row, COL);
                  -- increment variables
                  Col Count := Col Count + 1;
                              := Col_Indices'SUCC(COL);
                  COL
                             := Row Indices'SUCC(S Row);
                  S Row
              end loop Subtract Elements From Zero;
           -- add one to the diagonal element
          Answer(Row, COL) := Answer(Row, S Col) + 1.0;
          exit Do Every Row when Row Count = Input'LENGTH(1);
          Row Count := Row Count + 1;
```

```
:= Row Indices'SUCC(Row);
                            := Col Indices' SUCC(S Col);
                 S Col
             end loop Do Every Row;
      else
          raise Dimension Error;
      end if;
      return Answer;
   end Subtract From Identity;
pragma PAGE;
   function "+" (Left : Matrices;
                   Right: Matrices) return Matrices is
      -- declaration section
      Answer : Matrices(Left'range(1), Left'range(2));
      Col Count : POSITIVE;
      Row Count : POSITIVE;
      L Col
               : Col Indices;
      L Row : Row Indices;
R Col : Col Indices;
R Row : Row Indices;
S Col : Col Indices;
      S_Row : Row_Indices;
__ ______
-- -- begin function "+"
   begin
      -- make sure both input matrices are square matrices of the same size
      if Left'LENGTH(1) = Left'LENGTH(2) and
          Left'LENGTH(1) = Right'LENGTH(1) and
         Right'LENGTH(1) = Right'LENGTH(2) then
          -- addition calculations will only be carried out on the bottom half
          -- of the input matrices followed by assignments to the symmetric
         -- elements in the top half of the matrix
         Row Count := 1;
         -- as L Row goes down the rows, S Col will go across the columns
         L Row := Left'FIRST(1);
                    := Left'FIRST(2);
          S Col
         R Row
                     := Right'FIRST(1);
         Do All Rows:
             loop
```

```
Col Count := 1;
                 -- as L Col goes across the columns, S Row will go down the rows
                 L Col := Left'FIRST(2);
                 S Row := Left'FIRST(1);
                 R Col := Right'FIRST(2);
                 Add Bottom Half Elements:
                     TOOD
                        Answer(L_Row, L_Col) := Left(L_Row, L_Col) +
                                                   Right(\overline{R} Row, \overline{R} Col);
                        -- exit when diagonal element has been reached
                        exit Add Bottom Half Elements when Col Count = Row Count;
                        -- assign value to symmetric element in top half of matrix
                        -- (do this after exit since diagonal elements don't have
                        -- a corresponding symmetric element)
                        Answer(S Row, S Col) := Answer(L Row, L Col);
                        -- increment values
                        Col Count := Col Count + 1;
                        L_Col := Col_Indices'SUCC(L_Col);
S_Row := Row_Indices'SUCC(S_Row);
R_Col := Col_Indices'SUCC(R_Col);
                     end loop Add_Nottom_Half_Elements;
                 exit Do All Rows when Row Count = Left'LENGTH(1);
                 Row Count := Row Count + T;
                 L_Row := Row_Indices'SUCC(L Row);
                           := Col Indices'SUCC(S Col);
                 S_Col
                 R Row := Row Indices' SUCC(R Row);
             end loop Do All Rows;
      else
          raise Dimension Error;
      end if;
      return Answer;
   end "+":
pragma PAGE;
   function "-" (Left : Matrices;
                  Right: Matrices) return Matrices is
      -- declaration section
       _____
      Answer : Matrices(Left'range(1), Left'range(2));
```

```
Col Count : POSITIVE;
      Row Count : POSITIVE;
      L Col
                  : Col Indices;
      L Row : Row Indices;
R Col : Col Indices;
R Row : Row Indices;
S Col : Col Indices;
S Row : Row Indices;
-- -- begin function "+"
  ______
   begin
       -- make sure both input matrices are square matrices of the same size
       if Left'LENGTH(1) = Left'LENGTH(2) and
          Left'LENGTH(1) = Right'LENGTH(1) and
          Right'LENGTH(1) = Right'LENGTH(2) then
          -- addition calculations will only be carried out on the bottom half
          -- of the input matrices followed by assignments to the symmetric
          -- elements in the top half of the matrix
          Row Count := 1;
          -- as L Row goes down the rows, S Col will go across the columns
          L Row
                     := Left'FIRST(1);
          S Col
                       := Left'FIRST(2);
                       := Right'FIRST(1);
          R Row
          Do All Rows:
              loop
                 Col Count := 1;
                 -- as L Col goes across the columns, S Row will go down the rows
                 L Col := Left'FIRST(2);
                 S Row := Left'FIRST(1);
                 R Col := Right'FIRST(2);
                 Add Bottom Half Elements:
                     loop
                         Answer(L Row, L Col) := Left(L Row, L Col) -
                                                      Right(\overline{R} Row, \overline{R} Col);
                         -- exit when diagonal element has been reached
                         exit Add Bottom Half Elements when Col Count = Row Count;
                         -- assign value to symmetric element in top half of matrix
                         -- (do this after exit since diagonal elements don't have
                         -- a corresponding symmetric element)
                         Answer(S Row, S Col) := Answer(L Row, L Col);
                         -- increment values
                         Col Count := Col Count + 1;
```

```
L Col
                               := Col Indices'SUCC(L Col);
                     S_Row
R_Col
                               := Row_Indices'SUCC(S_Row);
                               := Col Indices'SUCC(R Col);
                  end loop Add Bottom Half Elements;
               exit Do All Rows when Row Count = Left'LENGTH(1);
               Row Count := Row Count + \overline{1};
                     := Row Indices' SUCC(L Row);
               L Row
                       := Col_Indices'SUCC(S_Col);
               S_Col
               R_Row := Row_Indices'SUCC(R_Row);
           end loop Do All Rows;
      else
         raise Dimension Error;
     end if;
      return Answer;
  end "-";
end Symmetric_Full_Storage_Matrix_Operations Unconstrained;
```

```
100
```

```
separate (General Vector Matrix Algebra)
package body Diagonal Matrix Operations is
-- -- local declarations
   type Col Markers is array(Col Indices) of POSITIVE;
   type Row Markers is array(Row Indices) of POSITIVE;
   Col Marker : Col Markers;
   Row Marker: Row Markers;
   Row Minus Col Indices Pos First : constant INTEGER
                                      := Row Indices'POS(Row Indices'FIRST) -
                                         Col Indices'POS(Col Indices'FIRST);
   Local Identity Matrix : constant Diagonal Matrices := (others => 1.0);
   Local Zero Matrix
                         : constant Diagonal Matrices := (others => 0.0);
pragma PAGE;
   function Identity_Matrix return Diagonal Matrices is
   begin
      return Local Identity Matrix;
   end Identity Matrix;
pragma PAGE;
   function Zero Matrix return Diagonal Matrices is
   begin
      return Local Zero Matrix;
   end Zero Matrix;
pragma PAGE;
   procedure Change Element (New Value : in
                                                 Elements;
                                     : in
                                                 Row Indices;
                              Row
                                                 Col Indices;
                              COL
                                        : in
                             Matrix : out Diagonal Matrices) is
  begin
      -- make sure element referenced is on the diagonal
      if Row Marker(Row) = Col Marker(COL) then
         Matrix(Row Marker(Row)) := New Value;
      else
         raise Invalid Index;
      end if;
```

```
end Change Element;
pragma PAGE;
   function Retrieve Element (Matrix : Diagonal Matrices;
                                Row
                                       : Row Indices;
                                         : Col Indices) return Elements is
                                COL
   begin
      -- make sure (row,col) falls on the diagonal
      if Row Marker(Row) /= Col Marker(COL) then
          raise Invalid Index;
      end if;
      return Matrix(Row Marker(Row));
   end Retrieve Element;
pragma PAGE;
   function Row_Slice (Matrix : Diagonal_Matrices;
                               : Row Indices) return Row Slices is
                        Row
      -- declaration section
      Col Spot : Col Indices;
      Answer : Row Slices;
-- -- begin function Row Slice
   begin
      -- zero out slice
      Answer := (others => 0.0);
      -- insert diagonal element
      Col Spot := Col_Indices'VAL(Row_Indices'POS(Row) -
                                    Row Minus Col Indices Pos First);
      Answer(Col Spot) := Matrix(Row Marker(Row));
      return Answer;
   end Row_Slice;
pragma PAGE;
   function Column_Slice (Matrix : Diagonal_Matrices;
                           COL : Col Indices) return Col Slices is
      -- declaration section
      Answer : Col Slices;
      Row Spot : Row Indices;
```

```
__ ______
-- -- begin function Column Slice
   begin
      -- zero out answer and then insert diagonal value
      Answer := (others => 0.0);
      -- insert diagonal value
      Row Spot := Row Indices'VAL(Col Indices'POS(COL) +
                                    Row Minus Col Indices Pos First);
      Answer(Row Spot) := Matrix(Col Marker(COL));
      return Answer;
   end Column Slice;
pragma PAGE;
   function Add To Identity (Input : Diagonal Matrices)
                             return Diagonal Matrices is
      -- declaration section
      -----
      Answer : Diagonal Matrices;
-- -- begin function Add to Identity
   begin
      Process:
         for INDEX in 1.. Entry Count loop
            Answer(INDEX) := Input(INDEX) + 1.0;
         end loop Process;
      return Answer;
   end Add_To_Identity;
pragma PAGE;
   function Subtract From Identity (Input : Diagonal Matrices)
                                     return Diagonal Matrices is
      -- declaration section
      Answer : Diagonal_Matrices;
-- -- begin function Subtract From Identity
```

```
begin
      Process:
         for INDEX in 1.. Entry Count loop
            Answer(INDEX) := 1.0 - Input(INDEX);
         end loop Process;
      return Answer;
   end Subtract From Identity;
pragma PAGE;
   function "+" (Left : Diagonal Matrices;
                Right : Diagonal Matrices) return Diagonal Matrices is
      -- declaration section
      -----
      Answer : Diagonal Matrices;
-- -- begin function "+"
  begin
      Process:
         for INDEX in 1.. Entry Count loop
            Answer(INDEX) := Left(INDEX) + Right(INDEX);
         end loop Process;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Left : Diagonal Matrices;
                Right: Diagonal Matrices) return Diagonal Matrices is
      -- declaration section
      ______
     Answer : Diagonal Matrices;
-- -- begin function "-"
  begin
     Process:
         for INDEX in 1.. Entry Count loop
            Answer(INDEX) := Left(INDEX) - Right(INDEX);
         end loop Process;
```

```
return Answer;
   end "-";
pragma PAGE;
-- begin processing for Diagonal
-- Matrix Operations package
begin
   Init Block:
       declare
          Col Count : POSITIVE;
          Row Count : POSITIVE;
       begin
          -- make sure lengths of indices are the same
          if Row Slices'LENGTH = Col Slices'LENGTH then
             -- initialize row and column marker arrays
             Row_Count := 1;
             Row Init:
                 for Row in Row Indices loop
                    Row_Marker(Row) := Row_Count;
                                     := Row Count + 1;
                    Row Count
               end loop Row Init;
             Col Count := 1;
             Col Init:
                 for COL in Col Indices loop
                    Col_Marker(COL) := Col_Count;
                    Col Count
                                    := Col_Count + 1;
                 end loop Col Init;
          else
             raise Dimension Error;
          end if;
      end Init Block;
end Diagonal Matrix Operations;
```



```
separate (General Vector Matrix Algebra)
package body Vector Scalar Operations Unconstrained is
pragma PAGE;
   function "*" (Vector : Vectors2;
                 Multiplier : Scalars) return Vectors1 is
      -- declaration section-
      _____
      Answer : Vectors1(Indices1'FIRST ...
                         Indices1'VAL(Vector'LENGTH-1 +
                                       Indices1'POS(Indices1'FIRST) ));
      A Index : Indices1:
      V Index : Indices2;
-- -- begin function "*"
__ _____
   begin
     A Index := Indices1'FIRST:
     V Index := Indices2'FIRST;
     Process:
        loop
           Answer(A Index) := Vector(V Index) * Multiplier;
           exit Process when V Index = Vector'LAST;
           A Index := Indices1\(^{\text{SUCC}}(A \) Index);
           V Index := Indices2'SUCC(V Index);
        end loop Process;
     return Answer;
   end "*";
pragma PAGE;
   function "/" (Vector : Vectors1;
                 Divisor: Scalars) return Vectors2 is
      -- declaration section-
      _____
      Answer : Vectors2(Indices2'FIRST ...
                         Indices2'VAL(Vector'LENGTH-1 +
                                       Indices2'POS(Indices2'FIRST) ));
      A Index : Indices2;
      V Index : Indices1;
-- -- begin function Vector Scalar Divide
```

```
begin

A_Index := Indices2'FIRST;
V_Index := Indices1'FIRST;
Process:
    loop

Answer(A_Index) := Vector(V_Index) / Divisor;
    exit Process when V_Index = Indices1'LAST;
    A_Index := Indices2'SUCC(A_Index);
    V_Index := Indices1'SUCC(V_Index);
    end loop Process;
    return Answer;
end "/";
end Vector Scalar Operations Unconstrained;
```



```
separate (General Vector Matrix Algebra)
package body Matrix Scalar Operations Unconstrained is
pragma PAGE;
   function "*" (Matrix : Matrices1;
                 Multiplier: Scalars) return Matrices2 is
__
     -- declaration section-
     _______
     Answer : Matrices2
                  (Row Indices2'FIRST ...
                   Row Indices2'VAL(Matrix'LENGTH(1)-1 +
                                    Row Indices2'POS(Row Indices2'FIRST) ),
                   Col Indices2'FIRST ...
                   Col Indices2'VAL(Matrix'LENGTH(2)-1 +
                                    Col Indices2'POS(Col Indices2'FIRST) ));
     A Col : Col Indices2;
     A_Row : Row_Indices2;
M_Col : Col_Indices1;
     M Row : Row Indices1;
-- -- begin function "*"
__ _____
  begin
     A Row := Row Indices2'FIRST;
     M Row := Matrix'FIRST(1);
     Row Loop:
        Ioop
           A Col := Col Indices2'FIRST;
           M Col := Matrix'FIRST(2);
           Col Loop:
                loop
                  Answer(A Row, A Col) := Matrix(M Row, M Col) * Multiplier;
                  exit Col Loop when M Col = Matrix'LAST(2);
                  A Col := Col Indices2'SUCC(A Col);
                  M_Col := Col_Indices1'SUCC(M_Col);
               end loop Col Loop;
           exit Row Loop when M Row = Matrix'LAST(1);
           A Row := Row Indices 2'SUCC(A Row);
           M Row := Row Indices1'SUCC(M Row);
        end loop Row_Loop;
     return Answer;
  end "*";
```

```
pragma PAGE;
   function "/" (Matrix : Matrices2;
                  Divisor: Scalars) return Matrices1 is
      -- declaration section-
      Answer : Matrices1
                  (Row Indices1'FIRST ...
                   Row Indices1'VAL(Matrix'LENGTH(1)-1 +
                                     Row Indices1'POS(Row Indices1'FIRST) ),
                   Col Indices1'FIRST ..
                   Col Indices1'VAL(Matrix'LENGTH(2)-1 +
                                     Col Indices1'POS(Col Indices1'FIRST) ));
      A_Col : Col_Indices1;
A_Row : Row_Indices1;
      M Col : Col Indices2;
      M Row : Row Indices2;
-- -- begin function "/"
   begin
      A Row := Row Indices1'FIRST;
      M Row := Matrix'FIRST(1);
      Row Loop:
         Ioop
            A Col := Col Indices1'FIRST;
            M Col := Matrix'FIRST(2);
            Col Loop:
                 loop
                   Answer(A_Row, A_Col) := Matrix(M_Row, M_Col) / Divisor;
                   exit Col Loop when M Col = Matrix'LAST(2);
                   A Col := Col Indices I'SUCC(A Col);
                   M Col := Col Indices2'SUCC(M Col);
               end loop Col_Loop;
            exit Row Loop when M Row = Matrix'LAST(1);
            A_Row := Row_Indices1'SUCC(A_Row);
            M_Row := Row_Indices2'SUCC(M_Row);
         end loop Row Loop;
      return Answer;
   end "/";
end Matrix_Scalar_Operations_Unconstrained;
```

```
separate (General Vector Matrix Algebra)
package body Diagonal Matrix Scalar Operations is
pragma PAGE;
   function "*" (Matrix : Diagonal Matrices1;
                Multiplier : Scalars) return Diagonal Matrices2 is
   - -----
     -- declaration section-
     -----
     Answer : Diagonal Matrices2;
     Index1 : Diagonal Rangel;
     Index2 : Diagonal Range2;
-- -- begin function "*"-
   begin
     Index1 := Diagonal Range1'FIRST;
     Index2 := Diagonal Range2'FIRST;
     Process:
        loop
           Answer(Index2) := Matrix(Index1) * Multiplier;
           exit Process when Index1 = Diagonal Range1'LAST;
           Index1 := Diagonal Range1'SUCC(Index1);
           Index2 := Diagonal Range2'SUCC(Index2);
        end loop Process;
     return Answer;
  end "*";
pragma PAGE;
  function "/" (Matrix : Diagonal Matrices2;
                Divisor: Scalars) return Diagonal Matrices1 is
     ______
     -- declaration section-
     Answer : Diagonal Matrices1;
     Index1 : Diagonal Rangel;
     Index2 : Diagonal Range2;
-- -- begin function "/"-
  begin
     Index1 := Diagonal Range1'FIRST;
```

Index2 := Diagonal Range2'FIRST;

```
pra
```

```
Process:
         loop
             Answer(Index1) := Matrix(Index2) / Divisor;
             exit Process when Index1 = Diagonal Range1'LAST;
             Index1 := Diagonal_Range1'SUCC(Index1);
             Index2 := Diagonal Range2'SUCC(Index2);
         end loop Process;
      return Answer;
   end "/";
pragma PAGE;
-- begin processing for package body
begin
   -- make sure instantiated diagonal matrices are of the same size
   if Diagonal_Matrices1'LENGTH /= Diagonal Matrices2'LENGTH then
      raise Dimension Error;
   end if;
```

end Diagonal Matrix Scalar Operations;

```
separate (General Vector Matrix Algebra)
package body Matrix Matrix Multiply Unrestricted is
pragma PAGE;
   function "*" (Left : Left Matrices;
                  Right: Right Matrices) return Output Matrices is
      -- declaration section-
      Answer : Output Matrices;
M_Answer : Output_Row_Indices;
M_Left : Left_Row_Indices;
N_Left : Left_Col_Indices;
N_Right : Right_Row_Indices;
P_Answer : Output_Col_Indices;
P_Right : Right_Col_Indices;
-- -- begin of function "*"
   begin
      M Answer := Output Row Indices'FIRST;
      M Left := Left Row Indices'FIRST;
      M Loop:
          loop
             P Answer := Output Col Indices'FIRST;
             P Right := Right Col Indices'FIRST;
             P Loop:
                 loop
                     Answer(M Answer, P Answer) := 0.0;
                     N Left
                                                    := Left Col Indices'FIRST:
                    N Right
                                                     := Right Row Indices'FIRST;
                     N_Loop:
                        loop
                            Answer(M Answer, P Answer) :=
                                Answer(M Answer, P Answer) +
                                Left(M_Left, N_Left) * Right(N_Right, P_Right);
                            exit N Loop when N Left = Left Col Indices'LAST;
                            N Left := Left Col Indices'SUCC(N Left);
                            N Right := Right Row Indices'SUCC(N Right);
                        end loop N Loop;
                     exit P Loop when P Right = Right Col_Indices'LAST;
                    P Right := Right Col Indices'SUCC(P Right);
                     P Answer := Output Col Indices'SUCC(P Answer);
                 end loop P Loop;
```

```
exit M Loop when M Left = Left Row Indices'LAST;
               M Left := Left Row Indices'SUCC(M Left);
               M Answer := Output Row Indices'SUCC(M Answer);
         end loop M Loop;
      return Answer;
   end "*":
pragma PAGE;
-- begin processing for package body
begin
-- -- make sure dimensions are compatible; to be compatible the following
-- -- conditions must'exist:
-- -- must be trying to multiply: [m \times n] \times [n \times p] := [m \times p]
   -- "n's"
                                                                            -- "m's"
           Right Matrices'LENGTH(2) = Output Matrices'LENGTH(2)) then
                                                                            -- "p's"
      -- dimensions are incompatible
      raise Dimension Error;
   end if;
```



end Matrix Matrix Multiply_Unrestricted;



```
separate (General Vector Matrix Algebra)
package body Matrix Vector Multiply Unrestricted is
pragma PAGE;
   function "*" (Matrix : Input Matrices;
                 Vector: Input Vectors) return Output Vectors is
      -----
      -- declaration section-
      Answer : Output Vectors;
      M Answer : Output Vector Indices;
      M Matrix : Row Indices;
      N Matrix : Col Indices;
      N_Vector : Input_Vector_Indices;
-- -----
-- -- begin function "*"
   begin
      M Answer := Output Vector Indices'FIRST;
      M_Matrix := Row_Indices'FIRST;
      M Loop:
         loop
            Answer(M Answer) := 0.0;
            N Matrix := Col Indices'FIRST;
            N Vector := Input Vector Indices'FIRST;
            N Loop:
               loop
                  Answer(M Answer) := Answer(M Answer) +
                               Matrix(M Matrix, N Matrix) * Vector(N Vector);
                  exit N Loop when N Matrix = Col Indices'LAST;
                  N Matrix := Col Indices'SUCC(N Matrix);
                  N Vector := Input Vector Indices'SUCC(N Vector);
               end loop N_Loop;
               exit M Loop when M Matrix = Row Indices'LAST;
               M Matrix := Row Indices'SUCC(M Matrix);
               M_Answer := Output_Vector_Indices'SUCC(M Answer);
         end loop M Loop;
      return Answer;
   end "*";
pragma PAGE;
-- begin processing for package body
```



begin



```
separate (General Vector Matrix Algebra)
package body Vector Vector Transpose Multiply Unrestricted is
pragma PAGE;
   function "*" (Left : Left Vectors ;
               Right: Right Vectors) return Matrices is
      -- declaration section
      _____
      Answer : Matrices;
      M Answer : Row_Indices;
      M Left : Left Vector Indices;
      N Answer : Col Indices;
      N Right : Right Vector Indices;
-- -----
-- -- begin function "*"
-----
   begin
      M Answer := Row Indices'FIRST;
      M Left := Left Vector Indices'FIRST;
      M Loop:
         loop
            N Right := Right Vector Indices'FIRST;
           N Answer := Col Indices'FIRST;
           N Loop:
              loop
                 Answer(M_Answer, N_Answer) := Left(M_Left) * Right(N_Right);
                 exit N Loop when N Right = Right Vector Indices'LAST;
                 N Right := Right Vector Indices'SUCC(N Right);
                 N Answer := Col Indices'SUCC(N Answer);
              end loop N Loop;
            exit M Loop when M Answer = Row Indices'LAST;
           M Answer := Row Indices'SUCC(M Answer);
           M_Left := Left_Vector_Indices'SUCC(M_Left);
      end loop M Loop;
      return Answer;
  end "*";
pragma PAGE;
-- begin processing for package body
```

```
3,53
```



```
separate (General Vector Matrix Algebra)
package body Matrix Matrix Transpose Multiply Unrestricted is
pragma PAGE;
   function "*" (Left : Left Matrices;
                 Right: Right Matrices) return Output Matrices is
      -- declaration section
      -----
      Answer : Output Matrices;
      M Answer : Output Row Indices;
      M_Left : Left_Row_Indices;
N_Left : Left_Col_Indices;
N_Right : Right_Col_Indices;
      P Answer: Output Col Indices;
      P Right : Right Row Indices;
__ _____
-- --- begin function "*"
   begin
      M Answer := Output Row Indices'FIRST:
      M Left := Left Row Indices'FIRST;
      M Loop:
         loop
            P Answer := Output Col Indices'FIRST;
            P_Right := Right_Row_Indices'FIRST;
            P Loop:
               loop
                  Answer(M_Answer, P_Answer) := 0.0;
                  N Left := Left Col Indices'FIRST;
                  N Right := Right Col Indices'FIRST;
                  N Loop:
                     loop
                        Answer(M Answer, P Answer) :=
                           Answer(M Answer, P Answer) +
                           Left(M Left, N Left) * Right(P Right, N Right);
                        exit N Loop when N Left = Left Col Indices'LAST;
                        N Left := Left Col Indices'SUCC(N Left);
                        N Right := Right Col Indices'SUCC(N Right);
                     end loop N Loop;
                  exit P Loop when P Answer = Output Col Indices'LAST;
                  P_Answer := Output_Col_Indices'SUCC(P_Answer);
                  P Right := Right Row Indices'SUCC(P Right);
               end loop P Loop;
```

```
exit M Loop when M Answer = Output Row Indices'LAST;
             M Answer := Output Row Indices'SUCC(M Answer);
             M Left := Left Row Indices'SUCC(M Left);
          end loop M_Loop;
      return Answer;
   end "*";
pragma PAGE;
-- begin processing for package body
begin
-- -- make sure dimension are compatible
-- -- need to have: [m \times n] \times [p \times n] := [m \times p]
   if not (Left Matrices'LENGTH(1) = Output Matrices'LENGTH(1) and
                                                                             -- "m's"
            Left_Matrices'LENGTH(2) = Output_Matrices'LENGTH(2) and -- "n's"
            Right Matrices'LENGTH(1) = Output Matrices'LENGTH(2)) then -- "p's"
      raise Dimension Error;
   end if;
end Matrix Matrix Transpose Multiply Unrestricted;
```



```
separate (General Vector Matrix Algebra)
package body Dot Product Operations Unrestricted is
pragma PAGE;
   function Dot Product (Left : Left Vectors;
                          Right: Right Vectors) return Result Elements is
      -- declaration section-
      ______
      Answer : Result Elements;
      L Index : Left Indices;
      R Index : Right Indices;
-- -- begin function Dot Product-
   begin
      Answer := 0.0;
      L Index := Left Indices'FIRST;
      R Index := Right Indices'FIRST;
      Process:
         loop
            Answer := Answer + Left(L_Index) * Right(R_Index);
            exit Process when L Index = Left Indices'LAST;
            L Index := Left Indices'SUCC(L Index);
            R Index := Right Indices'SUCC(R Index);
         end loop Process;
      return Answer;
   end Dot Product;
pragma PAGE;
-- begin processing for package body
begin
-- -- make sure instantiated vectors are of the same length
   if Left Vectors'LENGTH /= Right Vectors'LENGTH then
      raise Dimension Error;
   end if;
end Dot Product Operations Unrestricted;
```

(

```
separate (General Vector Matrix Algebra)
package body Diagonal Full Matrix Add Unrestricted is
pragma PAGE:
   function "+" (D Matrix : Diagonal Matrices;
                  F Matrix: Full Input Matrices) return Full Output Matrices is
      -- declaration section-
      Answer : Full_Output_Matrices;
      A Col Index : Full Output Col Indices;
      A Col Marker : Full Output Col Indices;
      A_Row_Index : Full_Output_Row_Indices;
D_Index : Diagonal_Range;
F_Col_Index : Full_Input_Col_Indices;
      F Row Index : Full Input Row Indices;
-- -- begin function "+"
-- -----
   begin
      -- first assign a row full of values, then add in diagonal element
      A Col Marker := Full Output Col Indices'FIRST;
      A_Row_Index := Full_Output_Row_Indices'FIRST;
      D Index
                := Diagonal Range'FIRST;
      F_Row_Index := Full Input Row Indices'FIRST;
      Add Loop:
         Toop
             A Col Index := Full Output Col Indices'FIRST;
             F Col Index := Full Input Col Indices'FIRST;
             Assign Loop:
                loop
                    Answer(A Row Index, A Col Index) :=
                       F Matrix(F Row Index, F Col Index);
                    exit Assign Loop
                       when A Col Index = Full Output Col Indices'LAST;
                    A Col Index := Full Output Col Indices'SUCC(A Col Index);
                    F Col Index := Full Input Col Indices'SUCC(F Col Index);
                end loop Assign Loop;
             Answer(A Row Index, A Col Marker) :=
                Answer(A Row Index, A Col Marker) + D Matrix(D Index);
             exit Add_Loop when D_Index = Diagonal_Range'LAST;
            A Col Marker := Full Output Col Indices'SUCC(A Col Marker);
A Row Index := Full Output Row Indices'SUCC(A Row Index);
             D Index := D Index + 1;
             F Row Index := Full Input Row Indices'SUCC(F Row Index);
```

THIS REPORT HAS BEEN DELIMITED AND CLEARED FOR PUBLIC RELEASE UNDER DOD DIRECTIVE 5200.20 AND NO RESTRICTIONS ARE IMPOSED UPON ITS USE AND DISCLOSURE.

DISTRIBUTION STATEMENT A

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

```
separate (General Vector Matrix Algebra)
package body Vector Operations Constrained is
pragma PAGE;
   function "+" (Left : Vectors;
                Right: Vectors) return Vectors is
      -- declaration section-
      ______
      Answer : Vectors;
-- -- begin function "+"
   begin
     Process:
         for INDEX in Indices loop
           Answer(INDEX) := Left(INDEX) + Right(INDEX);
        end loop Process;
     return Answer;
   end "+";
pragma PAGE;
   function "-" (Left : Vectors;
                Right: Vectors) return Vectors is
     -- declaration section-
     -----
     Answer : Vectors;
-- -- begin function "-"
  begin
     Process:
        for INDEX in Indices loop
           Answer(INDEX) := Left(INDEX) - Right(INDEX);
        end loop Process;
     return Answer;
  end "-";
```

```
pragma PAGE;
   function Vector Length (Input: Vectors) return Vector Elements is
     -- declaration section-
           : Vector Elements Squared;
  -- begin function Vector Length
__ _____
   begin
     Temp := 0.0;
     Process:
        for INDEX in Indices loop
           Temp := Temp +
                   Input(INDEX) * Input(INDEX);
        end loop Process;
     return Sqrt(Temp);
  end Vector Length;
pragma PAGE;
   function Dot Product (Left : Vectors;
                        Right: Vectors) return Vector Elements Squared is
     -- declaration section
     Answer : Vector_Elements_Squared;
-- -- begin function Dot Product
-- -----
  begin
     Answer := 0.0;
     Process:
        for INDEX in Indices loop
           Answer := Answer + Left(INDEX) * Right(INDEX);
        end loop Process;
     return Answer;
  end Dot Product;
end Vector Operations Constrained;
```

```
separate (General Vector Matrix Algebra)
package body Matrix Operations Constrained is
pragma PAGE;
   function "+" (Left : Matrices;
                 Right: Matrices) return Matrices is
      -- declaration section-
      Answer : Matrices;
-- -- begin function "+"
   begin
      Row Loop:
         for Row in Row Indices loop
            Col Loop:
               For COL in Col Indices loop
                  Answer(Row, COL) := Left(Row, COL) +
                                      Right(Row, COL);
               end loop Col_Loop;
         end loop Row Loop;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Left : Matrices;
                 Right: Matrices) return Matrices is
     -- declaration section-
     Answer : Matrices;
-- -- begin function "-"
-- -----
  begin
     Row Loop:
        for Row in Row_Indices loop
            Col Loop:
               for COL in Col Indices loop
```

```
Answer(Row, COL) := Left(Row, COL) -
                                     kight(Row, COL);
              end loop Col Loop;
         end loop Row Loop;
      return Answer;
   end "-";
pragma PAGE;
   function "+" (Matrix : Matrices;
                Addend : Elements) return Matrices is
      -- declaration section-
      -----
      Answer : Matrices;
-- -- begin function "+"
__ ____
   begin
      Row Loop:
        for Row in Row Indices loop
           Col Loop:
              For COL in Col Indices loop
                 Answer(Row, COL) := Matrix(Row, COL) + Addend;
              end loop Col Loop;
        end loop Row Loop;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Matrix : Matrices;
                Subtrahend: Elements) return Matrices is
     -- declaration section-
      -----
     Answer : Matrices;
-- -- begin function "-"
  begin
     Row Loop:
```





```
for Row in Row Indices loop
             Col Loop:
                for COL in Col_Indices loop
                   Answer(Row, COL) := Matrix(Row, COL) - Subtrahend;
                end loop Col Loop;
          end loop Row Loop;
      return Answer:
   end "-":
pragma PAGE;
   procedure Set To Identity Matrix (Matrix: out Matrices) is
      -- declaration section
      _____
      COL : Col Indices;
      Row : Row Indices;
-- -- begin procedure Set To Identity Matrix
   begin
      -- make sure input matrix is a square matrix
      if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
         Matrix := (others => (others => 0.0));
         COL := Col Indices'FIRST;
         Row := Row Indices'FIRST;
         Row Loop:
            TOOD
                -- set diagonal element equal to 1
               Matrix(Row, COL) := 1.0;
               exit when Row = Row Indices LAST;
               COL := Col_Indices'SUCC(COL);
               Row := Row_Indices'SUCC(Row);
            end loop Row_Loop;
      else
         -- do not have a square matrix
         raise Dimension Error;
      end if;
   end Set_To_Identity_Matrix;
pragma PAGE;
   procedure Set_To_Zero_Matrix (Matrix : out Matrices) is
```

```
begin
```

```
Matrix := (others => (others => 0.0));
end Set_To_Zero_Matrix;
end Matrix_Operations_Constrained;
```

```
separate (General Vector Matrix Algebra)
package body Dynamically_Sparse_Matrix_Operations_Constrained is
pragma PAGE:
   procedure Set To Identity Matrix (Matrix: out Matrices) is
      -- declaration section
      COL : Col Indices;
      Rcw : Row Indices;
-- -- begin procedure Set to Identity Matrix
   begin
       -- make sure input matrix is a square matrix
       if Matrix'LENGTH(1) = Matrix'LENGTH(2) then
          Matrix := (others => (others => 0.0));
          COL := Col Indices'FIRST;
          Row := Row Indices' FIRST;
          Row Loop:
             Ioop
                -- set diagonal element equal to 1.0
                Matrix(Row, COL) := 1.0;
                exit when Row = Row Indices'LAST;
                COL := Col_Indices'\(\overline{S}\)UCC(COL);
                Row := Row Indices' SUCC(Row);
             end loop Row_Loop;
      else
          raise Dimension Error;
      end if;
   end Set To Identity Matrix;
pragma PAGE;
   procedure Set To Zero Matrix (Matrix: out Matrices) is
   begin
      Matrix := (others => (others => 0.0));
   end Set To Zero Matrix;
pragma PAGE;
   function Add_To_Identity (Input : Matrices) return Matrices is
```

```
______
      -- declaration section
      -----
      Answer : Matrices:
      COL : Col_Indices;
Row : Row_Indices;
 -- -- begin function Add to Identity
_______
   begin
      -- make sure input is a square matrix
      if Input'LENGTH(1) = Input'LENGTH(2) then
         Answer := Input;
         -- add "identity" values to diagonal elements
         COL := Col_Indices'FIRST;
         Row := Row Indices'FIRST;
         Row Loop:
            Toop
                if Answer(Row, COL) /= 0.0 then
                   Answer(Row, COL) := Answer(Row, COL) + 1.0;
                else
                   Answer(Row, COL) := 1.0;
               end if;
               exit when Row = Row Indices'LAST;
               COL := Col Indices'SUCC(COL);
               Row := Row Indices' SUCC(Row);
             end loop Row Loop;
      else
         raise Dimension Error;
      end if;
      return Answer;
   end Add To Identity;
pragma PAGE;
   function Subtract From Identity (Input: Matrices) return Matrices is
      -- declaration section
      Answer : Matrices;
      COL : Col_Indices;
```

```
: Row Indices;
       Row
-- -- begin procedure Subtract From Identity
   begin
       -- make sure input is a square matrix
       if Input'LENGTH(1) = Input'LENGTH(2) then
          COL := Col Indices'FIRST;
          Row := Row Indices'FIRST;
          Row Loop: loop
                Col Loop:
                    for Temp Col in Col Indices loop
                       if Input(Row, Temp_Col) /= 0.0 then
                          Answer(Row, Temp_Col) := - Input(Row, Temp_Col);
                          Answer(Row, Temp Col) := 0.0;
                       end if;
                    end loop Col_Loop;
                if Answer(Row, COL) /= 0.0 then
                   Answer(Row, COL) := Answer(Row, COL) + 1.0;
                   Answer(Row, COL) := 1.0;
                end if;
                exit when Row = Row Indices'LAST;
                COL := Col_Indices'SUCC(COL);
                Row := Row_Indices'SUCC(Row);
             end loop Row Loop;
      else
          raise Dimension Error;
      end if;
      return Answer;
   end Subtract From Identity;
pragma PAGE;
   function "+" (Left : Matrices;
                  Right: Matrices) return Matrices is
      -- declaration section
      Answer : Matrices;
```

```
-- -- begin function "+"
   begin
      Row Loop:
         for Row in Row Indices loop
            Col Loop:
               For COL in Col Indices loop
                  if Left(Row, COL) = 0.0 then
                     if Right(Row, COL) = 0.0 then
                        Answer(Row, COL) := 0.0;
                     else
                        Answer(Row, COL) := Right(Row, COL);
                     end if;
                  elsif Right(Row, COL) = 0.0 then
                     Answer(Row, COL) := Left(Row, COL);
                     Answer(Row, COL) := Left(Row, COL) +
                                         Right(Row, COL);
                  end if;
               end loop Col Loop; .
         end loop Row Loop;
      return Answer;
   end "+";
pragma PAGE;
   function "-" (Left : Matrices;
                 Right: Matrices) return Matrices is
      ------
     -- declaration section
     -----
     Answer : Matrices;
-- -- begin function "-"
  begin
     Row Loop:
         for Row in Row Indices loop
            Col Loop:
               for COL in Col Indices loop
                  if Left(Row, COL) = 0.0 then
                     if Right(Row, COL) = 0.0 then
```

```
separate (General Vector Matrix Algebra)
package body Symmetric Full Storage Matrix Operations Constrained is
pragma PAGE;
   Row : in Row_Indices;
COL : in Col Indices;
                            Matrix : in out Matrices) is
     -- declaration section-
     S Col : Col Indices;
     S_Row : Row_Indices;
-- -- begin procedure Change Element-
 - -----
  begin
     S Col := Col Indices'VAL(Row Indices'POS(Row) -
                              Row_Indices'POS(Row_Indices'FIRST) +
                              Col_Indices'POS(Col_Indices'FIRST));
     S Row := Row Indices' VAL(Col Indices' POS(COL) -
                              Col_Indices'POS(Col_Indices'FIRST) +
Row_Indices'POS(Row_Indices'FIRST));
     Matrix(Row, COL) := New Value;
     Matrix(S Row, S Col) := New Value;
  end Change Element;
pragma PAGE;
  procedure Set_To_Identity_Matrix (Matrix : out Matrices) is
     -- declaration section
     COL: Col Indices;
     Row : Row Indices;
- -- begin procedure Set to Identity Matrix
begin
     Matrix := (others => (others => 0.0));
     COL := Col Indices'FIRST;
     Row := Row Indices'FIRST;
     Row Loop:
        Toop
```

```
-- set diagonal element equal to
            Matrix(Row, COL) := 1.0;
            exit when Row = Row Indices'LAST;
            COL := Col Indices'SUCC(COL);
            Row := Row Indices'SUCC(Row);
         end loop Row Loop;
   end Set To Identity Matrix;
pragma PAGE;
   procedure Set To Zero Matrix (Matrix: out Matrices) is
   begin
      Matrix := (others => (others => 0.0));
   end Set_To_Zero_Matrix;
pragma PAGE;
   function Add To Identity (Input: Matrices) return Matrices is
      -- declaration section
     Answer : Matrices;
      COL : Col Indices;
Row : Row_Indices;
-- -- begin function Add to Identity
   begin
      Answer := Input;
      COL := Col Indices'FIRST;
      Row := Row Indices'FIRST;
      Access Diagonal Elements:
         loop
            Answer(Row, COL) := Answer(Row, COL) + 1.0;
            exit when Row = Row Indices'LAST;
            COL := Col Indices'SUCC(COL);
            Row := Row Indices'SUCC(Row);
         end loop Access Diagonal Elements;
      return Answer:
  end Add To Identity;
```

```
pragma PAGE;
   function Subtract From Identity (Input: Matrices) return Matrices is
       -- declaration section
      Answer : Matrices;
Row : Row_Indices;
S_Col : Col_Indices;
S_Row : Row_Indices;

    -- begin function Subtract from Identity

   begin
      -- handle first diagonal element
      Answer(Row Indices'FIRST, Col Indices'FIRST) :=
          1.0 - Input(Row_Indices'FIRST, Col_Indices'FIRST);
      -- will subtract the remaining of the input matrix from an identity matrix
      -- by doing the following:
      -- o subtracting the nondiagonal elements in the bottom half of the
      --
               matrix from 0.0,
      -- o assigning values obtained in the bottom half of the matrix to the
               symmetric elements in the top half of the matrix, and then
      -- o subtracting the diagonal elements from 1.0
      -- S Col will go across the columns as Row goes down the rows to keep
      -- track of the column containing the diagonal element
      S Col := Col Indices'SUCC(Col Indices'FIRST);
            := Row Indices'SUCC(Row Indices'FIRST);
      Do_Every_Row_Except_First:
          loop
              -- S Row will go down the rows as Col goes across the columns
              S Row := Row Indices'FIRST;
              Subtract Nondiagonal Elements From Zero:
                 for COL in Col Indices'FIRST ...
                               Col Indices' VAL(Row Indices' POS(Row) - 1) loop
                     Answer(Row,COL) := - Input(Row,COL);
                     Answer(S_Row, S_Col) := Answer(Row, COL);
                     S Row
                                 := Row Indices'SUCC(S Row);
                 end loop Subtract Nondiagonal Elements From Zero;
              -- subtract diagonal element from 1.0
             Answer(Row, S Col) := 1.0 - Input(Row, S Col);
              exit when Row = Row Indices'LAST;
              S Col := Col Indices'SUCC(S Col);
```

```
Row
                     := Row Indices'SUCC(Row);
          end loop Do Every Row Except First;
      return Answer;
   end Subtract From Identity;
pragma PAGE;
   function "+" (Left : Matrices;
                   Right: Matrices) return Matrices is
      -- declaration section
      ______
      Answer : Matrices;
      Row : Row Indices;
S Col : Col Indices;
S Row : Row Indices;
   -- begin function "+"
   begin
      -- handle first diagonal element
      Answer(Row_Indices'FIRST, Col_Indices'FIRST) :=
          Left(Row Indices'FIRST, Col Indices'FIRST) +
          Right(Row Indices'FIRST, Col Indices'FIRST);
      -- addition calculations will only be carried out on the bottom half
      -- of the input matrices followed by assignments to the symmetric
      -- elements in the top half of the matrix
      -- as Row goes down the rows, S Col will go across the columns to keep
      -- track of the column containing the diagonal element
      S Col := Col Indices'SUCC(Col Indices'FIRST);
      Row := Row Indices'SUCC(Row Indices'FIRST);
      Do All Rows Except First:
          loop
             -- as Col goes across the columns, S Row will go down the rows;
             S Row := Row Indices'FIRST;
             Add Bottom Half Elements:
                 for COL in Col Indices'FIRST ..
                              Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop
                    -- add elements in bottom half of the matrix
                    Answer(Row, COL) := Left(Row, COL) + Right(Row, COL);
                    -- assign value to symmetric element in top half of matrix
                    Answer(S_Row, S_Col) := Answer(Row, COL);
                                := Row_Indices'SUCC(S_Row);
                    S_Row
```

```
end loop Add Bottom Half Elements;
              -- add diagonal elements together
              Answer(Row, S Col) := Left(Row, S Col) + Right(Row, S Col);
              exit when Row = Row Indices'LAST;
              S Col := Col Indices'SUCC(S Col);
              Row := Row Indices'SUCC(Row);
          end loop Do_All_Rows_Except_First;
       return Answer:
   end "+";
pragma PAGE;
   function "-" (Left : Matrices;
                    Right: Matrices) return Matrices is
       -- declaration section
      Answer : Matrices;
Row : Row Indices;
S_Col : Col_Indices;
S_Row : Row_Indices;
-- -- begin function "-"
   begin
       -- handle first diagonal element
       Answer(Row Indices'FIRST, Col Indices'FIRST) :=
          Left(Row Indices'FIRST, Col Indices'FIRST) -
          Right(Row Indices'FIRST, Col Indices'FIRST);
      -- subtraction calculations will only be carried out on the bottom half
       -- of the input matrices followed by assignments to the symmetric
       -- elements in the top half of the matrix
       -- as Row goes down the rows, S Col will go across the columns to keep
       -- track of the column containing the diagonal element
       S Col := Col Indices'SUCC(Col Indices'FIRST);
      Row := Row Indices'SUCC(Row Indices'FIRST);
      Do All Rows Except First:
          loop
              -- as Col goes across the columns, S Row will go down the rows;
              S Row := Row Indices'FIRST;
              Subtract_Bottom_Half_Elements:
                 for COL in Col Indices'FIRST ..
                               Col_Indices'VAL(Row_Indices'POS(Row) - 1) loop
                     -- subtract elements in bottom half of the matrix
```

```
Answer(Row, COL) := Left(Row, COL) - Right(Row, COL);
                    -- assign value to symmetric element in top half of matrix
                    Answer(S_Row,S_Col) := Answer(Row, COL);
                    S Row
                               := Row Indices'SUCC(S_Row);
                end loop Subtract Bottom Half Elements;
             -- subtract diagonal elements together
             Answer(Row, S Col) := Left(Row, S Col) - Right(Row, S Col);
             exit when Row = Row Indices'LAST;
             S Col := Col Indices'SUCC(S Col);
             Row := Row_Indices'SUCC(Row);
          end loop Do_All_Rows_Except_First;
      return Answer;
   end "-";
pragma PAGE;
-- processing for Symmetric Full Storage
-- Matrix_Operations_Constrained package body
begin
   if Matrices'LENGTH(1) /= Matrices'LENGTH(2) then
      raise Dimension Error;
   end if;
end Symmetric Full Storage Matrix Operations Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
package body Vector Scalar Operations Constrained is
pragma PAGE;
   function "*" (Vector : Vectors2;
                Multiplier: Scalars) return Vectors1 is
     -- declaration section-
      ______
     Answer : Vectors1;
-- -- begin function "*"
  ------
   begin
    Process:
       for INDEX in Indices loop
          Answer(INDEX) := Vector(INDEX) * Multiplier;
       end loop Process;
    return Answer;
   end "*";
pragma PAGE;
   function "/" (Vector : Vectors1;
                Divisor : Scalars) return Vectors2 is
     -----
     -- declaration section-
     -----
     Answer : Vectors2;
-- -- begin function Vector Scalar Divide
  begin
        for INDEX in Indices loop
           Answer(INDEX) := Vector(INDEX) / Divisor;
        end loop Process;
     return Answer;
  end "/";
```

end Vector_Scalar_Operations_Constrained;



```
separate (General Vector Matrix Algebra)
package body Matrix Scalar Operations Constrained is
pragma PAGE;
   function "*" (Matrix : Matrices1;
                 Multiplier : Scalars) return Matrices2 is
     -- declaration section-
      ______
      Answer : Matrices2;
-- -- begin function "*"
  -----
   begin
      Row Loop:
         for Row in Row_Indices loop
            Col Loop:
                for COL in Col Indices loop
                  Answer(Row, COL) := Matrix(Row, COL) * Multiplier;
               end loop Col Loop;
         end loop Row Loop;
      return Answer;
   end "*";
pragma PAGE;
   function "/" (Matrix : Matrices2;
                Divisor : Scalars) return Matrices1 is
      -- declaration section-
      Answer : Matrices1;
-- -- begin function "/"
   begin
      Row Loop:
         For Row in Row Indices loop
            Col Loop:
                for COL in Col Indices loop
```

```
Answer(Row, COL) := Matlix(Row, COL) / Divisor;
end loop Col_Loop;
end loop Row_Loop;
return Answer;
end "/";
end Matrix_Scalar_Operations_Constrained;
```

```
separate (General_Vector_Matrix_Algebra)
function Matrix Matrix Multiply Restricted (Left : Left Matrices;
                 Right: Right Matrices) return Output Matrices is
-- -- declaration section-
   Answer
               : Output_Matrices;
-- begin of function Matrix Matrix Multiply Restricted
begin
   Answer := (others => (others => 0.0));
   M Loop:
      for M in M Indices loop
         P Loop:
             for P in P Indices loop
                N Loop:
                   for N in N Indices loop
                      Answer(M, P) := Answer(M, P) +
                                       Left(M, N) * Right(N, P);
                   end loop N_Loop;
             end loop P Loop;
      end loop M Loop;
   return Answer;
end Matrix_Matrix_Multiply_Restricted;
```

end Matrix_Vector_Multiply_Restricted;

```
separate (General_Vector_Matrix_Algebra)
function Matrix_Vector_Multiply_Restricted (Matrix : Input_Matrices;
                Vector: Input Vectors) return Output Vectors is
 - -- declaration section-
   Answer : Output_Vectors;
-- begin function Matrix Vector Multiply Restricted
begin
   Answer := (others => 0.0);
   M Loop:
      for M in Indices1 loop
          N Loop:
             for N in Indices2 loop
                 Answer(M) := Answer(M) +
                               Matrix(M, N) * Vector(N);
             end loop N Loop;
      end loop M Loop;
   return Answer;
```



```
separate (General Vector Matrix_Algebra)
function Vector Vector Transpose Multiply Restricted
           (Left : Left Vectors;
            Right: Right Vectors) return Matrices is
-- -- declaration section
  Answer : Matrices;
-- begin function Vector Vector Transpose Multiply Restricted
______
begin
  M Loop:
     for M in Indices1 loop
        N Loop:
           for N in Indices2 loop
              Answer(M, N) := Left(M) * Right(N);
           end loop N Loop;
  end loop M_Loop;
  return Answer;
end Vector Vector Transpose Multiply Restricted;
```

```
separate (General Vector Matrix Algebra)
function Matrix Matrix Transpose Multiply_Restricted
            (Left : Left Matrices;
            Right: Right Matrices) return Output Matrices is
-- -- declaration section
-- -----
   Answer : Output Matrices;
-- begin function Matrix Matrix Transpose Multiply Restricted
begin
   Answer := (others => (others => 0.0));
   M Loop:
      for M in M Indices loop
         P Loop:
            for P in P_Indices loop
               N Loop:
                  for N in N Indices loop
                      Answer(M, P) := Answer(M, P) +
                                      Left(M, N) * Right(P, N);
                  end loop N Loop;
            end loop P Loop;
      end loop M Loop;
   return Answer;
end Matrix Matrix_Transpose_Multiply Restricted;
```

```
separate (General Vector Matrix Algebra)
function Diagonal Full Matrix Add Restricted
             (D Matrix : Diagonal Matrices;
              F Matrix : Full Matrices) return Full Matrices is

    -- declaration section-

   Answer : Full Matrices;
   Diag Index : Diagonal_Range;
   INDEX : Indices;
--- begin function Diagonal Full Matrix Add Restricted
begin
-- -- assign all values to answer and then add in diagonal elements
   Answer := F Matrix;
-- -- now add in diagonal elements
   Diag Index := Diagonal Range'FIRST;
   INDEX
              := Indices'FIRST;
   Add Loop:
      Toop
         Answer(INDEX, INDEX) := Answer(INDEX, INDEX) + D Matrix(Diag Index);
         exit when INDEX = Indices'LAST;
         Diag Index := Diagonal Range'SUCC(Diag Index);
         INDEX
                     := Indices'SUCC(INDEX);
      end loop Add Loop;
   return Answer;
end Diagonal_Full_Matrix_Add_Restricted;
```



```
separate (General Vector Matrix Algebra)
package body Vector Matrix Multiply Unrestricted is
   function "*" (Vector : Input Vectors;
                 Matrix : Input Matrices) return Output Vectors is
      -- declaration section-
      ------
      Answer : Output Vectors := (others => 0.0);
      M V : Input Vector Indices;
     NA : Output_Vector_Indices;
N : Col_Indices;
M : Row_Indices;
-- -- begin function "*"
 _ _____
   begin
      N A := Output Vector Indices'FIRST;
      N := Col Indices'FIRST;
      N Loop:
         loop
            M V := Input Vector Indices'FIRST;
            M := Row Indices'FIRST;
            M Loop:
               loop
                  Answer (N A) := Answer(N A) + Vector(M V) * Matrix(M, N);
                  exit when M = Row Indices'LAST;
                  M := Row Indices'SUCC(M);
                  M V := Input Vector Indices'SUCC(M V);
               end loop M Loop;
            exit when N = Col Indices'LAST;
            N := Col Indices'SUCC(N);
            N A := Output Vector Indices'SUCC(N A);
         end loop N Loop;
     return Answer:
  end "*";
pragma PAGE;
-- begin package Vector Matrix Multiply Unrestricted
```

```
separate (General_Vector_Matrix_Algebra)
function Vector Matrix Multiply Restricted
             (Vector : Input Vectors;
             Matrix: Input Matrices) return Output Vectors is
-- -- declaration section
   Answer : Output_Vectors := (others => 0.0);
-- begin function Vector Matrix Multiply Restricted
begin
   N Loop:
     for N in Indices2 loop
         M Loop:
            for M in Indices1 loop
                Answer(N) := Answer(N) + Vector(H) * Matrix(M,N);
            end loop M_Loop;
      end loop N_Loop;
   return Answer;
end Vector Matrix Multiply Restricted;
```

```
separate (General Vector Matrix_Algebra)
package body Aba Trans Dynam Sparse Matrix Sq Matrix is
   function Sparse_Left_Multiply(Left : A_Elements;
                                  Right: B_Elements ) return A_Elements is
      Answer : A Elements;
   begin
      if Left = 0.0 then
        Answer := 0.0;
      else
         Answer := Left * A Elements( Right );
      end if;
      return Answer;
   end Sparse Left Multiply;
   function Sparse_Right_Multiply( Left : A_Elements;
                                   Right: A Elements ) return C Elements is
      Answer : C Elements;
   begin
      if Right = 0.0 then
        Answer := 0.0;
      else
        Answer := C Elements( Left * Right );
      end if;
      return Answer;
  end Sparse Right Multiply;
   function Matrix Multiply is new Matrix Matrix Multiply Restricted
      ( Left Elements => A Elements,
       Right Elements => B Elements,
       Output Elements => A Elements,
       M Indices => M Indices,
       N Indices => N Indices,
P Indices => N Indices,
       Left_Matrices => A_Matrices,
       Right Matrices => B Matrices,
       Output Matrices => A Matrices,
       11 + 11
                        => Sparse Left Multiply );
  function Matrix Transpose Multiply is new
     Matrix Matrix Transpose Multiply Restricted
        ( Left Elements => A Elements,
          Right Elements => A Elements,
          Output Elements => C Elements,
          M_Indices => M_Indices,
          N Indices
P Indices
P Indices
P => M Indices,
          Left Matrices => A Matrices,
          Right Matrices => A Matrices,
          Output Matrices => C_Matrices,
```

```
11 * 11
                            => Sparse Right Multiply );
   pragma PAGE;
   function Aba Transpose( A : A Matrices;
                            B : B Matrices )
                                 return C Matrices is
      Intermediate : A Matrices;
      Answer : C Matrices;
   begin
      -----
     - multiply A * B -
      Intermediate := Matrix_Nultiply( Left => A,
                                        Right \Rightarrow B);
     - multiply AB * transpose of A -
      Answer := Matrix_Transpose_Multiply( Left => Intermediate,
                                            Right \Rightarrow A);
      return Answer;
   end Aba Transpose;
end Aba_Trans_Dynam_Sparse_Matrix_Sq_Matrix;
```

```
separate (General Vector Matrix Algebra)
package body Aba Trans_Vector_Sq_Matrix is
   function Multiply_Vm( Left : Vector_Elements;
                           Right : Matrix_Elements ) return Vector_Elements is
   begin
      return Left * Vector Elements( Right );
   end Multiply Vm;
   function Multiply_Vv( Left : Vector_Elements;
                           Right: Vector Elements ) return Scalars is
      return Scalars( Left ) * Scalars( Right );
   end Multiply Vv;
   function Vector_Matrix_Multiply is new Vector_Matrix_Multiply_Restricted
             ( Input Vector Elements => Vector Elements, Matrix Elements => Matrix Elements,
               Output Vector Elements => Vector Elements,
                           => Indices,
=> Indices,
rs => Vectors,
ces => Matrices,
ors => Vectors,
               Indices1
               Indices2
              Input_Vectors
Input_Matrices
Output_Vectors
                                       => Multiply Vm );
   function Vector_Vector_Multiply is new Dot_Product_Operations_Restricted
                ( Left_Elements => Vector_Elements,
  Right_Elements => Vector_Elements,
                  Result_Elements => Scalars,
                  Right Vectors => Vectors,
                                    => Multiply_Vv );
   pragma PAGE;
   function Aba Transpose( A : Vectors;
                             B: Matrices ) return Scalars is
      Partial Answer : Vectors;
      Answer : Scalars;
   begin
      - multiply A * B -
     Partial Answer := Vector Matrix Multiply( Vector => A,
                                                    Matrix => B);
```

```
-- - multiply AB * transpose of A -

Answer := Vector_Vector_Multiply( Left => Partial_Answer, Right => A );

return Answer;
end Aba_Transpose;
end Aba_Trans_Vector_Sq_Matrix;
```

```
separate (General Vector Matrix Algebra)
package body Aba Trans Vector Scalar is
-- -- Operators provided for instantiations -
   function Multiply Vs( Left : Vector Elements;
                           Right: Scalars) return Vector Elements is
       return Left * Vector Elements( Right );
   end Multiply Vs;
-- -- This operator is not used, but is required for the instantiation. -
-- -- It is "dummied" out to make it as small as possible.
   function Divide_Vs( Left : Vector_Elements;
                        Right : Scalars ) return Vector_Elements is
   begin
      return Left;
   end Divide Vs;
   function Multiply_Vv( Left : Vector_Elements;
                           Right: Vector Elements ) return Matrix Elements is
      return Matrix Elements( Left ) * Matrix Elements( Right );
   end Multiply_Vv;
  -- Instantiations for ABA transpose -
      ----------
   package Vs_Opns is new Vector_Scalar_Operations_Constrained
                 ( Elements1 => Vector_Elements,
                  Elements2 => Vector Elements,
                  Scalars => Scalars,
Indices => Indices,
Vectors1 => Vectors,
                  Vectors1
Vectors2 => Vectors,
"*" => Multiply_Vs,
"/" => Divide_Vs );
   use Vs Opns;
   function Vv Transpose Multiply is new
      Vector Vector Transpose Multiply Restricted
          ( Left_Vector_Elements => Vector_Elements,
            Right_Vector_Elements => Vector_Elements,
           Matrix Elements => Matrix Elements,
Indices1 => Indices,
            Indices2
                                   => Indices.
```

```
Left_Vectors
Right_Vectors
                              => Vectors,
=> Vectors,
=> Matrices,
           Matrices
                                => Multiply Vv );
   pragma PAGE;
   function Aba Transpose( A : Vectors;
                           B : Scalars ) return Matrices is
      Partial_Answer : Vectors;
      Answer : Matrices;
   begin
      _____
     - multiply A * B -
     -----
     Partial_Answer := A * B;
     - multiply AB * transpose of A -
     Answer := Vv_Transpose_Multiply( Left => Partial_Answer,
                                       Right => A );
      return Answer;
   end Aba Transpose;
end Aba_Trans_Vector_Scalar;
```

```
separate (General Vector Matrix Algebra)
package body Column Matrix Operations is
pragma PAGE;
   function Set_Diagonal_And_Subtract_From_Identity
                      : Vectors;
      ( Column
        Active Column: Indices ) return Column Matrices is
      Answer : Column Matrices;
   begin
      Answer.Col Vector
                           := Column;
                           := TRUE;
      Answer.Diagonal
      Answer.Active Column := Active Column;
     Range Loop:
         for INDEX in Indices loop
            Answer.Col Vector( INDEX ) := - Answer.Col Vector( INDEX );
         end loop Range Loop;
      Answer.Col_Vector(Active_Column) := Answer.Col_Vector(Active_Column) +
                                           1.0:
      return Answer;
   end Set Diagonal And Subtract From Identity;
pragma PAGE;
   function Aba Transpose( A : Column Matrices;
                           B: B Matrices ) return C Matrices is
                  : C Matrices;
     Answer
     Temp Vector : Vectors := A.Col Vector;
   begin
      if A.Diagonal then
         Temp Vector( A.Active Column ) := Temp Vector( A.Active Column ) - 1.0;
         M Loop:
            for Row in Indices loop
               P Loop:
                  for COL in Indices loop
                     Answer( Row, COL ) := C_Matrix_Elements(
                        Temp Vector( Row ) * Temp Vector( COL ) *
                        B( A.Active_Column, A.Active_Column )
                        Temp Vector( COL ) * B( A.Active Column, Row ) +
                        Temp Vector( Row ) * B( A.Active Column, COL ) +
                        B( Row, COL ) );
                  end loop P Loop;
            end loop M Loop;
     else
         M1 Loop:
            for Row in Indices loop
               P1 Loop:
                  for COL in Indices loop
                     Answer( Row, COL ) := C_Matrix_Elements(
                        A.Col Vector( Row ) * A.Col Vector( COL ) *
                        B( A.Active Column, A.Active Column ) );
                  end loop P1 Loop;
            end loop M1 Loop;
```

```
end if:
      return Answer;
   end Aba Transpose;
pragma PAGE:
   function Aba Symm Transpose( A : Column Matrices;
                                B : B_Matrīces ) return C_Matrices is
                 : C Matrices;
      Answer
                : Indices;
      LAST
      Temp Vector : Vectors := A.Col_Vector;
   begin
      LAST := Indices'LAST;
      if A.Diagonal then -- Diagonal value is 1 --
         Temp_Vector( A.Active_Column ) := Temp_Vector( A.Active_Column ) - 1.0;
         M Loop:
            for Row in Indices loop
               P Loop:
                  - Calculate values -
                  for COL in Row .. Indices'LAST loop
                     Answer( Row, COL ) := C_Matrix_Elements(
                        Temp Vector( Row ) * Temp Vector( COL ) *
                        B( A.Active Column, A.Active Column )
                        Temp_Vector( COL ) * B( A.Active_Column, Row ) +
                        Temp Vector( Row ) * B( A.Active Column, COL ) +
                        B( Row, COL ) );
                     - Assign calculated value to corresponding -
                     - lower triangular position -
                     Answer( COL, Row ) := Answer( Row, COL );
                  end loop P Loop;
            end loop M_Loop;
     else
                               -- diagonal value is 0 --
        M1 Loop:
            for Row in Indices loop
               P1 Loop:
                  - Calculate values -
                  for COL in Row .. Indices'LAST loop
                     Answer( Row, COL ) := C Matrix Elements(
                        A.Col_Vector( Row ) * A.Col_Vector( COL ) *
                        B( A.Active_Column, A.Active_Column ) );
                     ______
                     - Assign calculated value to corresponding -
                     - lower triangular position -
                     Answer( COL, Row ) := Answer( Row, COL );
                  end loop P1 Loop;
            end loop M1 Loop;
     end if:
                                 -- Diagonal value is 0 --
```



return Answer;

end Aba_Symm_Transpose;

end Column_Matrix_Operations;



(This page left intentionally blank.)

SUPPLEMENTARY

INFORMATION

DEPARTMENT OF THE AIR FORCE

WRIGHT LABORATORY (AFSC) EGLIN AIR FORCE BASE, FLORIDA, 32542-5434



REPLY TO ATTN OF:

MNOT

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center ATTN: DTIC/HAR (Mr William Bush)

Bldg 5, Cameron Station Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

Techr.ical Report Number	AD Number
1. 88-18-Vol-4 2. 88-18-Vol-5 3. 88-18-Vol-6	ADB 120 251 ADB 120 252 ADB 120 253
4 . 88-25-Vol-1 5 . 88-25-Vol-2	ADB 120 309 ADB 120 310
6. 88-62-Vol-1 7. 88-62-Vol-2 8. 88-62-Vol-3	ADB 129 568 ADB 129 569 ADB 129-570
9 · 85-93-Vol-1 40 · 85-93-Vol-2 44 · 85-93-Vol-3	ADB 102-654 ADB 102-655 ADB 102-656
42. 88-18-Vol-1 15. 88-18-Vol-2 14. 88-18-Vol-7 15. 88-18-Vol-8 16. 88-18-Vol-9 17. 88-18-Vol-10 18.88-18-Vol-11 19. 88-18-Vol-12	ADB 120 248 ADB 120 249 ADB 120 254 ADB 120 255 ADB 120 256 ADB 120 257 ADB 120 258 ADB 120 259

2. If you have any questions regarding this request call me at DSN 872-4620.

Chief, Scientific and Technical

Information Branch

AFDTC/PA Ltr, dtd 30 Jan 92



DEPARTMENT OF THE AIR FORCE HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC) EQLIN AIR FORCE BASE, FLORIDA 32542-5000



REPLY TO ATTN OF:

PA (Jim Swinson, 882-3931)

30 January 1992

SUBJECT:

Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

VIRGINIA N. PRIBYLA, Lt Col, SAF

Chief of Public Affairs

AFDTC/PA 92-039